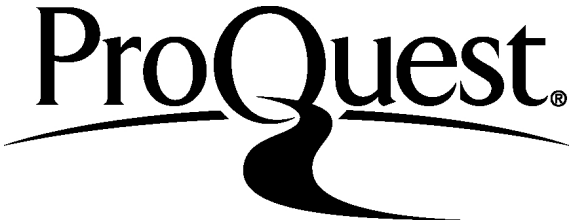
DO SOCIOLINGUISTIC VARIATIONS EXIST IN PROGRAMMING?

FARIHA NAZ
Bachelor of Computer Science and Information Technology, NED University of Engineering and Technology, 2010

A Thesis
Submitted to the School of Graduate Studies
of the University of Lethbridge
in Partial Fulfillment of the
Requirements for the Degree

MASTER OF SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA



DO SOCIOLINGUISTIC VARIATIONS EXIST IN PROGRAMMING?

FARIHA NAZ

Date of Defense: July 23, 2015

Dr. Jacqueline E. Rice Supervisor	Associate Professor	Ph.D.
Dr. Kevin Grant Committee Member	Associate Professor	Ph.D.
Dr. Javid Sadr Committee Member	Assistant Professor	Ph.D.
Dr. Inge Genee Committee Member	Associate Professor	Ph.D.
Dr. Howard Cheng Chair, Thesis Examination Committee	Associate Professor	Ph.D.

Abstract

Machine learning techniques are currently widely used in the analysis of natural language. This thesis focuses on extending these techniques for analysis of programming languages. In particular we are interested in determining whether there are differences in the use of programming languages that might be associated with the authors' gender. There are currently few studies that address possible relationships between linguistics and programming. In this thesis we use computer programs as the samples in our dataset. These programs have been written using the C++ programming language. We also acquired sociolinguistic information about the programmers, with the focus especially on gender. We use machine learning and statistical techniques to identify patterns (in language use) that are consistent for male and female programmers. The results of numerous experiments are encouraging. We demonstrate that we can predict the gender of programmers with 71% accuracy and detect similarities or dissimilarities in their programming style.

Acknowledgments

I am grateful to Dr. Jacqueline E. Rice for placing her belief and allowing me to explore this new research area. I would also like to express gratitude to Dr. Kevin Grant, Dr. Inge Genee, and Dr. Javid Sadr as my committee members for listening to my ideas and offering guidance. I felt like I was home due to the friendly, respectful, and caring behavior of the faculty members.

I am thankful to my fellow students for their continuous support throughout this journey. I would like to extend my gratitude towards many people inside and outside the university that have guided my way through this process and were there for me.

I am thankful for all of my family members including my mother Prof. Riaz-un-Nisa Syed (late), M. Rehman, Dr. Nauman, Sana, Basit, Bisma, M. Ali, Aisha, Imaan, and my extended family for their continuous guidance, love, and support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Human beings use "natural language" [35] such as English, French, and Urdu to communicate with each other. The use of natural language varies depending on the way people choose to behave within a society. The differences that influence language use mostly occur due to social factors such as age, region, ethnicity, socio-economic status (SES), and gender [31]. Socio-economic status includes education level, occupation, and income [35]. Similarly, in the society of programmers, differences in the utilization of a programming language may occur due to the above social factors as well.

Programming languages are designed for giving instructions to a computer (machine) in order to accomplish specific tasks [36]. A computer programmer (or software developer) writes programs using programming language in order to offer a solution to computational problems. For example, Excel^{TM} is software which can be utilized to carry out various tasks in order to perform mathematical functions. The syntax of a program is quite strictly determined by the programming language which is designed for facilitating instructions to a computer. Nonetheless, the choices left up to the programmer include the use of keywords, operators, statements, and comments. Thus, we used these choices to investigate the sociolinguistic variations in programming.

Computer programs are developed by programmers of different ages, genders, and ethnicities using various programming languages. For this reason, there may be the possibility of finding variations in how people use programming language in terms of the choices of keywords, operators, statements, and comments [39]. In this project we explore the pos-

sibility of identifying the gender of the computer program's author by applying machine learning techniques. We also examine possible statistical differences based on the usage of the above features to identify characteristics of male-written and female-written computer programs. Are there differences in the writing of computer programs that can be correlated with the gender of the programmers? To answer this question we start the investigation by considering male-written and female-written computer programs as our dataset.

Many problems that are related to the analysis and organization of data can be posed as machine learning problems. The kind of data being analyzed may vary. For instance, the data could be text documents written in natural language [22, 4] or computer programs written in Java [9]. We propose using several popular supervised learning techniques, including support vector machine (SVM), decision trees (J48), bayes classifier (naïve bayes), and nearest-neighbor (K*). The open-source implementations that we have chosen are the SVM^{light} and WEKA libraries, as they are mature libraries that are well-known in the machine-learning community. To perform statistical analysis we used SPSS [3].

SVM^{light} is an open-source implementation of the support vector machine algorithm developed by Joachims [23]. This algorithm is used to categorize various kinds of data belonging to the area of natural language and bioinformatics [4, 37]. WEKA [20] is also an open-source machine learning tool which is widely used. Previous uses include categorization of files stored on hard drives [42] and of programs written in Java [9]. WEKA is composed of various algorithms that can be used to preprocess data, extract features, and perform supervised or unsupervised learning of different kinds of data. We used SPSS for the statistical analysis of variations in features that could be used to identify characteristics of male-written and female-written C++ programs.

1.1 Motivation and Hypotheses

Misek-Falkoff [34] suggested that software linguistics is a domain in which techniques from linguistics can be applied to software applications. Each natural language has a *gram-*

mar which includes rules related to sentence structure, semantics, and lexicon. These rules are observed in the writing of text documents. Similarly, C++ is a programming language with its own *grammar* that allows specific syntax, semantics, and vocabulary to be utilized in the construction of computer programs. She suggested that techniques from linguistics can be used to analyze a program or a software package. One of the reasons for this is that in natural language a text is a collection of statement sequences. Similarly a computer program is a form of writing which qualified as a text. It is a collection of various sentences which is intended to be instructive or devised primarily to communicate. For this reason our first hypothesis is that the programs can be treated as text documents and the techniques that deal with text documents in natural language may be applicable to computer programs. For instance, to construct the numerical representation of computer programs we use term frequency and inverse document frequency (tf-idf) technique. This technique is popular for creating a numerical representation of text documents written in natural language [40, 46].

As first proposed by sociolinguist William Labov [31], social variability may influence linguistic variability. The linguistic variables are phonetic, morphosyntactic, lexical, and stylistic variables. The social variables that are studied by sociolinguists include socio-economic status (SES), age, ethnicity, region, and gender. Some researchers have examined these ideas in the area of natural language, in order to identify a specific social variable, that is, gender [4, 27]. In accordance with this, our second hypothesis, the social factor of gender may also correlate with variability in the writing and the development of C++ programs.

For this study we investigate whether we can determine if gender, as a social factor, correlates with the language use of computer programs. Gender was selected as the single social variable as a means of narrowing the scope of the study. This study builds on the work of Argamon et al., which is focused on the categorization of text documents on the basis of the author's gender [5, 4, 27]. In the future, we could include other variables such as age and socio-economic status as social variables.

1.2 Possible Contributions

This research may assist in identifying the gender of programmers using several machine learning algorithms. These algorithms include decision trees, naïve bayes, nearest neighbor, and support vector machine. Various researchers used these algorithms to analyze various kinds of data [22, 9, 42, 41]. The application of this work could include the following:

- The supervised learning models may aid in the automatic categorization of computer programs on the basis of the author's gender.
- The development of an Integrated Development Environment (IDE) could aid in improving the understanding of different programming styles by providing suggestions to male and female programmers.
- The knowledge about how different groups of programmers think and program differently could aid in the design of development teams.
- The knowledge about the use of programming languages could be used in the teaching of programming.

1.3 Organization of Thesis

In Chapter 2 we give an overview of sociolinguistics, machine learning, supervised learning, SVM^{light} and WEKA on the basis of the related research in these fields.

Chapter 3 provides the methodology that we apply to categorize female-written and male-written C++ programs. We explain the four main steps that we utilize in our methodology and describe all five experiments and their results. We also discuss threats to validity as well as the programming environment that we used.

Chapter 4 describes some analysis of features to identify gender-based variations.

In Chapter 5 we conclude this research and discuss possible future directions.

Chapter 2

Background and Literature Review

In this chapter we discuss related works that act as a foundation for this study in the area of sociolinguistics, authorship analysis (attribution), and machine learning.

2.1 Sociolinguistics

In a society a social identity is embodied in factors including speech, writing, clothing, commodities, and mannerisms. Each society imposes certain conventions on its members in terms of a person's occupation, age, gender, and socio-economic status (SES) [31, 35]. Wardhaugh [43] defines a society as "any group of people who are drawn together for a certain purpose or purposes", and a language as "what the members of a particular society speak". Within a society people belonging to different age, gender and SES groups (among others) are distinguishable based on the features of their oral communication including accent, grammar use and vocabulary [35]. The field of sociolinguistics involves the study of linguistic variation as it correlates with social factors which may influence the use of a language within a society [31, 35]. Linguistic variation that correlates with social factors is referred to as sociolinguistic variation.

The term sociolinguistics was coined by Currie [13] when he wrote about regional variation in the usage of the English language among Americans. However, sociolinguistic variation also occurs due to variables related to social class, gender, and age. The correlation between social and linguistic variation was first demonstrated by Fischer [15] which provided the basis for the sociolinguistic analysis of language. The language used by a

society is influenced by a number of factors. These factors in turn produce identifiable differences in oral or written communication.

Misek-Falkoff [34] explored the relationship between software and linguistics. As a result of her explorations she identified a new domain that she called "Software Linguistics". An analysis of natural language can be done by analyzing different phonetic, morphological, syntactic and semantic structures of a language on the basis of a set of rules which may be referred to as the grammar of a language [35]. For example, natural language analysis may be performed by examining each part of a word or how different words were grouped together within a sentence to express thoughts. Misek-Falkoff suggested that similar techniques were applicable towards the analysis of software code.

2.1.1 Sociolinguistic Variation

Sociolinguistics plays a critical role in analyzing differences in the use of natural language on the basis of social variables [31, 35]. The identification and interpretation of gender differences in a society of authors based on writing style has been explored by researchers [5, 4, 27] using techniques including machine learning and statistical analysis, as well as the combination of these approaches. Some researchers extended the field of identifying differences between authors of text documents towards the authors of computer programs.

In the case of determining various differences between authors of computer programs (members of the programming "society") similar or dissimilar patterns were identified on the basis of their programming styles [28]. A society of programmers is composed of people who are drawn together to develop software in order to solve a computational problem. Some variations may be found in computer programs that are written as part of software and there may be social variables that influence the programmer's style, just as society influences a person's verbal or written communication [39]. For instance, different ways to decompose or solve a particular computer programming problem might bring out some

variations, and differences in terms of identifier names used for methods and/or variables. Knowledge of these rules was used to demonstrate how groups use the same programming language to develop a software or computer program.

2.1.2 Related Work

2.1.3 Gender-Based Variations in Language

Argamon et al. [5] investigated gender differences in English literature as part of the large dataset known as the British National Corpus (BNC). The BNC is composed of both fiction and non-fiction texts including articles and books. The documents used in the study are articles/books that belong the fiction and non-fiction genres. The dataset used in [5] was composed of 604 documents and the average length of each document was more than 2,000 words. Each author contributed at most six documents as part of the dataset. The usage of lexical and syntactic features was analyzed according to the gender of authors. More than 1000 features were selected including a list containing 467 function words (e.g. *in, without, he, it, one, of, a*), and a list consisting of different sequences of words on the basis of parts-of-speech. A combination of machine learning and statistical analysis was employed to identify differences based on the authors' writing style. The machine learning techniques were applied to identify relevant features and statistical techniques were employed to test the significance of the gender-based differences in the frequency of features and their correlation with the two genres.

Argamon et al. [5] used a machine learning method named EG algorithm [32] to select a small set of the most useful features from 1000 features. The small list of features had an impact on the identification of textual documents on the basis of author's gender. As a result, only 50 features were identified by the EG algorithm that played a role in the differentiation between male-written and female-authored texts. The *Student's t-test* and *Mann-Whitney U test* statistical techniques were applied to identify differences in the usage of features based on the gender of authors. The feature frequencies were computed

per 10,000 tokens/words. In terms of frequency the distinguishing features of male texts were determiners and quantifiers, words that modify and precede nouns (e.g. *that, one, two, more*). In the documents written by men the determiners were more prevalent. The frequency mean of determiners was 2288, whereas in the documents authored by females the frequency mean for these words was 2060. In the female-written documents, the presence of pronouns (e.g. *I, you, myself, herself, she*) was dominant. The frequency mean of pronouns in female written documents was 1367, whereas in the male authored texts the frequency means of this type of words was 1142. The researchers observed that the differences between documents authored by males and females depend on the way ideas, people, and objects were presented. To investigate the features within a particular genre the correlation between male/female writing and non-fiction/fiction was computed using Pearson's Product Moment Coefficient [18]. Pearson's correlation demonstrated that there was a strong relationship between the characteristics found in documents written by male (female) and nonfiction (fiction) genres.

Argamon et al. [4] also explored gender differences using a total of 600 French literary and historical textual documents. In this dataset each author was represented by multiple documents. In this case, the machine learning algorithm support vector machine (SVM^{light} [22] as described briefly in section 2.3) was used to distinguish between male-written and female-written documents. The advantage of machine learning over statistical analysis of data was the construction of predictive model(s). Argamon et al. [4] created the support vector machine (SVM) model using labeled data samples of male-written and female-written documents. The support vector machine model was developed using SVM^{light} to discriminate between male-authored and female-authored documents on the basis of word distribution, usage, and their frequencies. The prevalent words and features present in the documents were identified using the support vectors from the SVM model. The model was able to predict the gender of the writers accurately by 90%. The results from [4] were similar to those from [5].

In much of the literature [22, 33] the categorization of textual documents was performed on the basis of the topic of documents; as a result the list of features used in a categorization was usually very large. However, Koppel et al. [27] used writing style to categorize text documents which resulted in the hand-selected set of features. The EG algorithm [26] was used as a learning method as well as a feature reduction algorithm for the gender-based categorization of the 566 documents that were part of the British National Corpus (BNC). The model created by the EG algorithm was able to achieve accuracy of approximately 80% when unseen textual documents were classified. The gender-based categorization of text documents written in English and French languages demonstrates that the different machine learning techniques can be utilized to analyze and classify the text documents of various natural languages.

The problem of categorizing text documents had also been pursued in the context of authorship analysis (or attribution) [27]. Authorship analysis is a process which distinguishes the author on the basis of their writing style. A feature was referred to as a "writer-specific feature" if it varied throughout the dataset but there was smaller (or no) variation in the writing of an individual author. The researchers assumed that over a period of time a writer developed consistent habits of composition style in terms of different features including word usage, sentence structures, phrase choices, and lexical categories [28]. Thus, author-based differences can be identified to analyze the authorship of text documents in natural language.

2.1.4 Authorship Attribution in Programs

The investigation of the divergence between the writings of different authors had been extended towards programming languages to demonstrate that an author of computer programs can also be identified. There were several differences in terms of feature usage that were associated with an author and can be extracted on the basis of coding style. Authorship attribution was applied towards the detection of plagiarism and the identification of

sources of computer viruses [28, 8]. Krsul and Sappford [28] classified computer programs to identify the author based on a set of features and coding style. The dataset used in the study was composed of programs written in C programming language. There 88 programs were collected from different problem domains. Both statistical and machine learning techniques were employed to identify the authors of the programs. The authors were recognized based on programmer-specific features. The set of features was found on the basis of the style and the structure of C programs. The statistical analysis of dataset was carried out using SAS [2] to identify feature-based differences. In addition, LNKnet software [29] was used to identify relevant features as well as to apply machine learning methods on the dataset. Thus, features were eliminated that contributed little to the classification of programs. Using LNKnet, various supervised and unsupervised learning algorithms were applied to the given dataset. For instance, neural network algorithms, nearest neighbor algorithms, k-means clustering, feature selection algorithms, and n-fold cross-validation (LOOCV) techniques were employed.

Some of the features examined to distinguish among programmers included the use and placement of brackets and comments; lengths of variable and comments; usage of data structures (or data type); and occurrences of “for” and “while” loops. The selection of variable names was also analyzed to investigate if a name represented the usage of a given variable. In addition to the analysis of features, the researchers also suggested that structural information can be acquired by visual analysis of computer programs. For instance, the use of blank lines was noticed because programmers used blank lines to separate independent blocks of code. These kinds of the repeated patterns were identified [28] in the construction of programs. Hence, the recognition of the authorship of C program was possible even with a limited dataset because programmers used style conventions with which they were familiar.

2.2 Machine Learning

With the advent of big data machine learning has become extremely popular with algorithms being used to analyze all manner of data, including text (e.g. document classification [22, 4]), biological data (e.g. genes and proteins [37]), and media (e.g facial recognition in images/video [41]). Machine learning algorithms are useful to acquire knowledge and interesting information from diverse types of data. Machine learning computer programs (or algorithms) automatically recognize patterns after learning from datasets in order to make decisions about future (or new) datasets.

Machine learning algorithms can be divided on the basis of how data can be dealt with in two particular approaches: *supervised* and *unsupervised* learning algorithms [18]. The assignment of datasets which is based on pre-defined class labels is called supervised learning (or classification). As an example, numeric values like $+1$ and -1 may represent two different classes. All data samples (or instances) that are present in the datasets may be associated with these class labels. However, in unsupervised learning (or clustering), datasets are not associated with predefined class labels. In clustering, datasets are divided into multiple groups to identify classes based on the similarity and dissimilarity among a set of data samples/tuples/instances that are present in datasets.

To utilize machine learning algorithms with textual data, in the field of text mining [42], there are four components: document representation, dimension (or feature) reduction, learning method, and testing protocol [39, 27]. Text mining is the procedure of identifying information and patterns from various types of textual data. The documents are represented as a vector which is composed of various features (dimensions) and their frequencies. Sometimes there are irrelevant features that may provide inaccurate results. The use of various feature reduction algorithms may be beneficial to reduce the dimensions by keeping only relevant features. The popular machine learning algorithms are employed as learning methods in order to construct models, which identify interesting patterns that can be used to accurately predict future (or unseen) datasets. As described in section 2.5, the

reliability of the machine learning models are tested using k-fold cross validation. The average values of evaluation metrics are collected in order to analyze the performance of models.

2.2.1 Supervised Learning (or Classification)

Supervised learning (or classification) is the process of determining a model that distinguishes data into different classes. The model is derived based on the analysis of a set of training data, that is, data for which class labels are known [18]. These models are called *classifiers*. The role of a classifier is to predict the categories of test data based on class labels. The classification of data can be performed using different machine learning algorithms directly or using their open-source implementation(s). The algorithms that can be applied to classify data include the support vector machine (SVM), naïve bayes, and the decision tree. The procedure of classification is a two-step process that includes

(i) learning (or training of a model), and

(ii) classification (or testing of a model).

In the *learning* step, a classifier is constructed on a predetermined set of classes and training data samples (or instances). The machine learning algorithms build a classifier (or model) by analyzing a training set in order to generalize useful information. In the *classification* step, the model is assessed by using test data samples (or instances) and their associated class labels [18]. Different evaluation metrics can be used to collect the results of the model on a given test dataset. These results give the percentage of the test data samples (or instances) that are correctly classified by the classifier. Supervised learning algorithms focus on the accuracy, precision, recall, and f-measure of a classifier (or model) for evaluation purposes. The values for these metrics range from 0% to 100%. The associated class labels of test data samples (or instances) are compared with the predictions produced by the model. If the results of the model (or classifier) are closer to 100%, then the model is considered useful to classify future (or unseen) data samples (or instances).

As an example, suppose a manager of some hypothetical store wants to classify a large set of items in the store based on the three kinds of responses to a sales campaign: good response, average response, and no response [18]. The model will be produced based on the features that are associated with the price and brand of an item. The resulting classification will distinguish each class from the others. This will demonstrate an organized picture of the given dataset. In this instance, the classification of the data would be beneficial to understand the impact of the sales campaign, and help to design a more effective campaign in the future on the basis of the above pre-defined features.

2.2.2 Data Transformation

Supervised learning algorithms require a particular data format [21] to perform the analysis of the dataset. In our study, the dataset was composed of computer programs that were treated as text documents. For this reason, the given dataset was transformed into the format required for classification algorithms using term frequency inverse document frequency (tf-idf) technique. The tf-idf is one of the techniques used in the area of automatic text retrieval [40] to create a numerical representation of textual documents in terms of features.

Term Frequency-Inverse Document Frequency (tf-idf) is a "vector-space model which represents an object as a vector of weighted indexing term, and define(s) object similarity in terms of those vectors." [46]. The tf-idf vector quantity has two parts: the tf part and the idf part. Term Frequency (tf) deals with one data sample and counts the number of times a specific word (or feature) appears in a data sample. Inverse Document Frequency (idf) counts the number of times the same word (or feature) has been seen in the entire dataset [46]. The tf-idf of a feature is the product of tf and idf. As a result, each textual document can be transformed into a numerical representation.

To perform supervised learning we used a dataset with two class labels. To illustrate the format of the dataset for a two-class learning problem, suppose that D is a complete dataset which consists of X_i and Y_i. We have $X_i \in \{X_1, X_2, \ldots, X_n\}$, where n is the total number

of tuples which are present in the dataset [18]. Each X_i is associated with the class label $Y_i \in \{+1, -1\}$. An individual tuple is composed of features, and a set of features which represents a single tuple is referred to as a feature vector. In the following section we discuss SVM^{light} and WEKA suite.

2.3 SVM^{light}

As described above, there are various machine learning algorithms that can be utilized to perform the classification of a given dataset. The state-of-the-art supervised learning method is the support vector machine (SVM), which is used in many applications including face detection [7], text categorization [4, 22], and searching for information on the internet [10]. For this reason, we also used the support vector machine algorithm as one of our four classification methods.

SVM^{light}, an open-source implementation of SVM [23], is used to categorize a two-class dataset; to apply standard kernel functions; to handle sparse representation; and to compute leave-one-out estimates of the precision and the recall evaluation metrics. The required format of the dataset is composed of tuples and class labels. Each tuple is represented by various features to create feature vectors as <feature>:<value>, where <feature> is the index of the feature and <value> is a numeric quantity.

The machine learning algorithm SVM is a kernel-based algorithm. The kernel function

$$K(x_i, x_j) = \varphi(x_i)^T \cdot \varphi(x_j)$$

calculates the dot product for the training vectors x_i in the possible high dimensional space [21]. In other words, training tuples (or data samples) of both linear (separable by a straight line) and nonlinear (not separable by a straight line) data can be mapped into the possible high dimensional space using kernel function.

There are four basic kernel functions that are accessible using SVM ^{light}: linear, poly-

nomial, radial basis function (RBF), and sigmoid kernel [21]. If a function draws a straight line to separate tuples (or data samples) of two classes, then this function is referred to as the "linear kernel function", and data is considered to be linearly separable. By default, the linear kernel is selected in SVM^{light}. If no straight line exists to separate classes, then data is linearly inseparable. The kernel functions which implement nonlinear mapping include the following: polynomial, radial basis function (RBF), and sigmoid kernel.

During the learning step, the choice of kernel plays a role in generalizing properly from the training tuples. In addition, the parameters of the SVM and the kernel need to be set in order to mitigate the risk of information loss and increase the performance [6]. Using SVM$^{\text{light}}$, the linear kernel function can be used if a dataset is composed of a large number of features [21]. The non linear kernel is suitable when the number of instances is larger than the number of features, so that features can be modeled in a higher dimensional space. In this work we have more instances than features. Thus, we used the radial basis kernel function:

$$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2),$$

where γ is the kernel parameter which should be greater than zero.

Furthermore, in the learning step, SVM searches for the maximum marginal hyperplane (MMH) which is the hyperplane with large distanced margin. A hyperplane that separates data tuples is defined as [18]:

$$W \cdot X + b = 0,$$

where $X = (X_1, X_2, \ldots, X_n)$ is a tuple consisting n attribute values, $W = W_1, W_2, \ldots, W_n$ is a weight vector, and b (bias) is a scalar. There is a possibility of finding various separating hyperplanes for a given training tuples. The hyperplane is created in the new dimension to separate the training tuples, on the basis of class labels. The hyperplane can be found using

support vectors and margins.

The distance between the hyperplane and the essential training tuples (support vectors) is useful to find the smallest or the largest distanced margins. Margins can be found on both sides of the hyperplane and continue to move away until the support vector of either class is found [18]. Generally, the margin with the larger distance on both sides of the hyperplane is accurate, because there will be less chance to misclassify during the prediction of class labels of unseen data in the classification step [45].

The training tuples that are present above or on the separating maximum margin hyperplane (MMH) are classified with one class label. Otherwise, data samples that fall below or on the (separating) MMH are labeled as another class. Using SVM$^{\text{light}}$ [21], the SVM parameter (C) can be employed to handle the trade-off between the margin and the learning error.

Support vectors are critical training tuples that give the most information about class labels Y_i. These tuples lie closest to the hyperplane and, for all i, satisfy the following equation [18]

$$Y_i(W \cdot X + b) \geqslant 1.$$

The SVM algorithm with even a small set of support vectors may give a good generalization of training data. The addition and removal of these tuples may have an impact on the classification of datasets in both learning and classification steps [18]. For instance, in the case of a dataset with two pre-defined class labels such as $+1$ and -1, if SVM finds data tuples (support vectors) above or on the margin of the hyperplane, then these data samples are classified as $+1$. Otherwise, data samples that fall below or on the margin of the hyperplane are labeled as -1. Hence, the goal of SVM is to search and find the "best" hyperplane using support vectors and margins to predict the test tuples more accurately.

2.4 WEKA

In addition to SVM^{light} we used Waikato Environment for Knowledge Analysis (WEKA) [20] which is an open-source machine learning software. WEKA is composed of various algorithms related to the preprocessing of data, feature selection, classification, clustering, and association rules. In this work, WEKA was used to perform the following tasks:

- to apply three filters to preprocess the given dataset,
- to use three supervised learning algorithms, and
- to apply two attribute selection algorithms to extract a small set of features.

2.4.1 Data Format

WEKA uses a particular representation of a dataset which is referred to as attribute-relation file format (ARFF). The ARFF file is composed of a matrix with rows and columns. The rows represent instances/tuples and the columns represent features/attributes [45]. The ARFF file format contains three tags:

(i) @relation

(ii) @attribute

(iii) @data

First is the header which gives information about the relation, which is in the format @relation <relation-name>; the second part provides information about the attributes (or features) and represented as @attribute <attribute-name> <datatype>, where datatype can be numeric (integer numbers), string (textual values) or nominal (list of possibilities); and the last part consists of instances (or tuples) [20, 45].

In some cases, the instances have features with zero value. To explicitly represent only nonzero features, a *sparse data* file would be a practical choice. A sparse data file is also

composed of the three tags as described above. However, the representation of instances is enclosed in a curly braces with the index number and value of nonzero attribute in the format: <index> <space> <value>, where <index> is the attribute (or feature) index starting from zero. In the sparse data files, features with zero values are not represented and it would be beneficial to visually analyze the features that are associated with the tuples and class labels in the given dataset. After the construction of datasets and the collection of results using various classification algorithms, comparative analysis may provide new insights about the dataset.

2.4.2 Data Preprocessing

To preprocess data, there are various filters present in WEKA such as NonSparse-ToSparse, Randomize, and Remove [45]. The *NonSparseToSparse* filter converts ARFF files into the data file which contains instances and their nonzero features. The *Randomized* filter is used to shuffle the order of the tuples/samples in the dataset. The *Remove* filter is useful to remove the features in order to create a dataset with relevant and useful features only.

2.4.3 Classification Algorithms

WEKA contains different types of machine learning methods but in this research we focused on the most widely used: classification (or supervised learning) algorithms. We developed three additional classifiers (or models) using the WEKA implementations of decision trees (J48), nearest-neighbor (K^*), and naïve bayes (NB) to categorize male-written and female-written computer programs. In the following subsections we briefly describe the above classification algorithms.

2.4.4 J48 Algorithm

C4.5 is an extension of ID3, a basic decision tree, which deals with numeric features, unavailable values, and prune decision trees [45]. WEKA contains an implementation of

C4.5 algorithm known as the J48 algorithm. A decision tree is constructed in a recursive way on the basis of features that partition the instances into distinct classes. A flow chart-like tree structure is called a decision tree. The topmost node is a root node; each internal node (non-leaf node) denotes a test on an attribute; each branch represents an outcome of the test; and each leaf node holds a class label [18]. All the internal nodes are denoted by rectangles and leaf nodes are denoted by ovals.

J48 algorithm works on a "divide-and-conquer" technique to learn from instances to identify the probability distribution of instances in the given dataset [45]. The probability distribution is identified by analyzing the class distribution and is usually stored in the leaf node of the tree. During the construction of a tree, gain ratio is used to determine the "best" attribute which serves as node in the decision tree [38, 17]. The test is carried out on the attribute to compare the numerical value of the attribute with a constant. The test is chosen that extracts the maximum amount of information from a set of instances.

To incorporate tests into the tree, two outcomes are required [45]. Each node must have a minimum of two (default value) instances which can be applied with the use of the *minNumObj* (M) parameter in WEKA. Each attribute can be used multiple times during the construction of a tree. To find a class label for a test instance, attribute values of that instance are tested against the decision tree. To classify the test tuple, a path is traced from the root to a leaf node. A tree can be pruned to remove irrelevant nodes from the decision tree [18, 45]. The pruning of the tree is controlled by the *confidenceFactor* (C) parameter. The default value of C is twenty-five percent and works well for most of the cases.

2.4.5 Naïve Bayes Algorithm

Naïve Bayes is a simple Bayesian classifier and commonly used to categorize documents for a two-class situation [27]. Bayesian classifier is a statistical classifier because it predicts the probability of a given instance that belongs to a particular class. Naïve Bayes works on the assumption that each attribute that is associated with a given class is indepen-

dent of other features. This is referred to as "class conditional independence" [25]. The dataset consisted of instances and their class labels. Each instance/tuple is represented in terms of the features/attributes. Naïve Bayes identifies the highest probability for a particular class label C_i, and chooses that label using the following formula [18, 45]:

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)},$$

where $P(C_i|X)$ is the probability that an instance X will belong to a particular class. The class with the highest probability is chosen [18]. $P(X|C_i)$ is the probability of X based on a specified class. $P(C_i)$ is the probability of instance that belongs to a particular class C_i. There are two class labels: male and female. $P(X)$ is the probability of an instance X observed and is often constant. The WEKA implementation of naïve bayes model is a "standard probabilistic nave bayes classifier" [45] which uses estimator classes [24]. We are interested in the use of a parameter which is *useKernelEstimator* (K). The reason for using this parameter is that we want to use the procedure which does not assume any particular distribution for the numeric features.

2.4.6 K* Algorithm

K* is used to perform the supervised learning of a dataset [42, 30] and is considered suitable for continuous feature values. This is a nearest-neighbor method and implemented as part of the WEKA tool as a "nearest neighbor with generalized distance function" [45]. There are various distance functions to compute the distance between instances. The distance was calculated between training and test instances by considering all possibilities. The K* classifier is based on the use of an entropic distance measure to provide prediction for future datasets. K* is also able to handle missing attribute values. In the WEKA implementation an important parameter of this algorithm is the blending parameter, which is the "sphere of influence" [11]. This sphere will demonstrate the number of important neighbor

instances.

Similar instances are identified using an entropic distance metric. The distance from the training sample is calculated. The new/test instance x is compared with the existing/training ones b_i using the distance metric [45]. Class labels are assigned to the new (test) instance on the basis of the closest k-nearest existing instances, b_i [19] as in the equation

$$K^*(b_i,x) = -\log P^*(b_i,x),$$

where $i \in \{1,2,...k\}$ and P is the probability of all possible paths from a training (b) to a test (x) instance.

2.4.7 Attribute Selection

Generally a dataset is represented by a set of features, and relevant features/attributes are usually unknown. The presence of irrelevant features degrades the predictive ability of models. For this reason, a small set of features should be chosen, consisting of features that are sufficient for learning and improving the quality of the concept description (model) [14, 5]. WEKA also contains various algorithms to identify useful features. To optimize the performance of model, numerous statistical and machine learning algorithms can be used to choose relevant features from a given dataset. The attribute selection process in WEKA is composed of two aspects [45]:

- an attribute evaluator, and
- a search method.

Attribute evaluator methods are used to extract relevant features from original features. The attribute (or feature) selection methods are heuristic procedures to evaluate the value of an attribute either as a subset or as an individual. These include *InfoGainAttributeEval* and *CfsSubsetEval* [45]. There are various search methods that work in combination with attribute evaluators to identify the "best" features. The attribute selection process aids in the

extraction of important features and enables one to identify irrelevant features which may confuse the machine learning system (or classification model). However, the reduction in the dimensionality (number of features) of a dataset may impact the performance of learning algorithms.

2.4.8 Information Gain

In this thesis the single-attribute evaluator *InfoGainAttributeEval* from WEKA was applied on the given dataset. *InfoGainAttributeEval* is a statistical technique which evaluates features on the basis of Information Gain. Information gain is calculated on the basis of the difference between the original information about the proportion of classes and the new information that is obtained after the identification of the useful attribute. The information gain is calculated using the following formula [18, 45]:

$$\text{InfoGain}(\text{Class}, \text{Attribute}) = \text{H}(\text{Class}) - \text{H}(\text{Class}|\text{Attribute})$$

InfoGainAttributeEval determines the amount of information about the class label that has been gained using an attribute. This evaluator was combined with the ranking method known as *Ranker* [45]. The ranking method constructs a list to show the rank of an individual attribute based on the value of Information Gain. The *Ranker* method enables us to identify features that can either be retained or discarded. As a result, a small number of features can be identified which may have an impact on the performance of the machine learning models.

2.4.9 Correlation Based Feature Subset Selection

Evaluating each attribute individually is faster but less accurate. For this reason, we also use an attribute subset evaluator, that is, *CfsSubsetEval* to select a subset of features [45]. The correlation-based feature subset selection (CFS) is a part of WEKA and calculates the predictive ability of individual attribute along with the redundancy of features within the

subset [17]. We used forward selection to begin with all the features. A subset of useful features evolves from the list of candidate subsets. The features within the subset are highly correlated with the class but have low correlation with each other. The *CfsSubsetEval* works with various standard search methods such as best-first search and genetic algorithm.

In this research we use the *Genetic* search method which is a simple genetic algorithm implemented as part of WEKA software. This search algorithm is analogous to "natural selection" [16], which holds the notion that the life of organisms is related, evolves naturally over time, and descendants may vary from ancestor. As a result, during the evolutionary process of the population of organisms only the fittest (best) ones survive. This method works in the following way, as described in [18].

The population of features is created to use for searching. This population consists of randomly generated rules. The rule is a string which is composed of bits representing features and classes. The rule of fitness, which is the notion that the fittest individual survives within the population, is evaluated on the basis of data tuples. As a result, a new population is developed which consists of the rules that have been accumulated from the survived individuals. The offspring of the population is created by the application of crossover and mutation, which are genetic operators. Crossover is the swapping of substrings from one rule (string) to another, in order to create new strings. Mutation is the process of inserting, deleting, rearranging, duplicating, and moving a selected bit in the rule (string). The new populations continue to be generated until a population evolves in which each rule (string) satisfies a rule of fitness. Using WEKA, the parameters can be tweaked [45]. The parameters include crossover probabilities, number of generations, population size, seed, starting point of attribute to create a population, number of generations and mutations.

2.5 Model Evaluation Techniques

In supervised learning there are two steps: learning and classification (as described in section 2.2.1). The common techniques to handle data in order to perform the supervised

learning are: hold out and cross validation methods.

2.5.1 Hold out Method

The hold out method is used to divide the entire dataset into two independent sets which are partitioned using fixed percentage of split [18]. One dataset is composed of two-thirds of the data samples/tuples that were used in the *learning* step, that is, in training and construction of the model. The other dataset consists of the remaining one-third of the tuples that are reserved for testing of the model (the *classification* step of the supervised learning [45, 18]). The test dataset should not be used during the learning step to reduce the chance of inaccurate predictions.

In the presence of scarce data the drawback of using the hold out method is that there is the possibility of overfitting or underfitting. Sometimes the knowledge from a given training dataset is not sufficient to accurately classify a test dataset. There is then a high risk that random predictions made by the model will work and be "learned"; that is, there is a risk of learning the incorrect information from the dataset. This can result when the model learns from the given training dataset but on the test dataset only 50% of the data was classified correctly. This problem is called overfitting [14]. The opposite of this problem is underfitting in which the model is biased because it misses important information in terms of the relevant features and class labels. Hence, the hold out method reduces the amount of data available to develop a model in the learning step and tests a model in the classification step. This can be mitigated by applying cross validation [14]. The k-fold cross validation technique is used in the thesis.

2.5.2 Cross Validation

Cross validation is a model evaluation technique in which a dataset is partitioned using a fixed number of folds (k) [18, 45]. As an example, 2-fold partition splits the dataset into two partitions and uses each partition for training and testing. This process is repeated two times so that during the test of a model each instance is used at least once.

Leave-one-out cross validation (LOOCV) is a special case and can be applied using WEKA and SVM^{light}. LOOCV is n-fold cross validation, where n=k and "n" is the total number of instances that are present in the dataset. There will be "n" turns in which each instance is left out of the training set and is then used as test set for validating a model. To further describe leave-one-out cross validation, suppose that there are *n* data tuples (or instances), then the training and testing needs to be carried out *n* times and the number of iterations will be k = n. For each iteration training of the model will be performed using n-1 data tuples. The model will be tested using the remaining data tuples (or instances). The averages of the model evaluation metrics of all runs (or iterations) is used as the final estimates of the predictive ability of developed models.

2.5.3 Evaluation Metrics

The predictive quality of a model (or classifier) is identified by analyzing various evaluation metrics such as accuracy, recall, precision, and f-measure [18, 20]. The score of each evaluation metric is computed using a confusion matrix which consists of values representing how many samples gave TP (true positives), TN (true negatives), FP (false positives), and FN (false negatives).

A confusion matrix demonstrates how well a model is able to classify the number of tuples (or instances) that are associated with various classes. The diagonal elements of the matrix show the instances that are classified correctly. For instance, the confusion matrix is a table of size "m x m", as shown in Table 2.1, where m is the number of classes. In this confusion matrix samples from female programmers are considered to be positive tuples, while those from male programmers are considered to be negative tuples [18]:

- true positives (TP) are positive tuples that the model correctly predicted as being positive tuples;
- true negatives (TN) are negative tuples that the model correctly predicted as being negative tuples;

- false positives (FP) are negative tuples that are misclassified by the model as positive tuples; and
- false negatives (FN) are positive tuples that are incorrectly labeled as negative tuples by the classifier.

Table 2.1: 2x2 Confusion Matrix.

Predicted Class				
Actual Class	GENDER	Female	Male	TOTAL
	Female	TP	FN	P
	Male	FP	TN	N
	TOTAL	P'	N'	P+N

Using a confusion matrix the accuracy, precision, recall and f-measure for the model can be calculated as described in [18, 9, 45]. Accuracy is defined as the percentage of instances that are labeled correctly by the classifier/model. This measure reflects the performance of the model in the recognition of instances that were associated with various classes. The accuracy of the model can be calculated by using the following formula [18]:

$$Accuracy = \frac{TP + TN}{P + N}$$

In supervised learning other measures can also be used to analyze the performance of the model [18]. Precision measures the percentage of positive data tuples that are classified as a specific class and actually belong to the specific class. A precision score of 100% (or 1) for a class demonstrates that every instance that labeled as that class does indeed belong to that class. It is the "measure of exactness" [18] and can be computed as:

$$Precision = \frac{TP}{TP + FP}$$

Recall measures the percentage of positive data tuples that are correctly classified as a specific class label; however, there is no information about data tuples (or instances) that are mislabeled by the model. A recall score of 100% (or 1) for a class demonstrates that every instance that is associated with the class is labeled as belonged to that class but without the information of other instances that are misclassified. It is the "measure of completeness" or "true positive rate" [18], as shown in the following formula:

$$Recall = \frac{TP}{TP+FN}, or$$

$$Recall = \frac{TP}{P}$$

Precision and recall are used widely in supervised learning [18, 42]. In some cases there tends to be an inverse relationship between precision and recall [18, 9]. Therefore, there is a possibility that sometimes models may achieve high precision but low recall. Many researchers in this field and in the area of information retrieval used f-measure to represent the combined measure of both precision and recall. F-measure represents the "harmonic mean of precision and recall" [18] as shown below [18, 45]:

$$F-measure = \frac{(2*Recall*Precision)}{(Recall+Precision)}, or$$

$$F-measure = \frac{2.TP}{2.TP+FP+FN}$$

As a result, researchers also used f-measure to evaluate the predictive ability of classification models [9, 18]. In the case of binary (two-class) classification problem, the average values of the evaluation metrics would be considered "best" if they lie near 100%.

Chapter 3

Methodology

In this chapter, we discuss the significant steps which we use to move forward towards the categorization of computer programs. We describe our experimental work and results. We also discuss potential threats towards the validity of this research. Additionally, we provide the overview of the programming environment used in this study to perform various experiments.

3.1 Experimental Work

To pursue the sociolinguistic analysis of computer programs we need to perform four main steps [27, 39]:

(i) create the dataset;

(ii) transform the dataset into a numerical representation;

(iii) apply machine learning method(s), specifically, supervised learning methods; and

(iv) apply testing protocol.

Our dataset consists of C++ computer programs collected from computer science courses that are offered at the University of Lethbridge. The numerical representation of a given dataset is produced by calculating the frequency of a set of features. The machine learning methods consist of various algorithms that are applied to the dataset to develop concept descriptions. The concept descriptions are referred to as models or classifiers. The machine learning methods include decision trees (J48), nearest neighbor (K^*), support vector

machine (SVM), and naïve bayes (NB). These perform the supervised learning of the computer programs to construct models. We then use cross-validation as a testing protocol to assess the models developed by the supervised learning methods.

In the early stages of this research work, we used an automated parts-of-speech (POS) tagger [1] on a C++ program (code sample) in an attempt to choose a set of attributes/features that might be useful to aid in the categorization. The POS tagger divided each line of a program into segments and identified tokens (separated by blank spaces). However, these segments and tokens were not a useful match to what were considered to be relevant segments and/or tokens in a programming language. This was not surprising as the POS tool generated labeling of each token which was based on parts-of-speech such as nouns, verbs, adverbs, and prepositions.

During the application of the POS tagger we ran into some issues. One problem with the usage of the POS tagger was that the data inside quotes was not tagged and tokenized correctly. Another issue was that the lexical category of datatype ("int") was mislabeled. Generally, the data type of a variable or function was employed to describe the type of data being stored. However, the POS tagger labeled "int" as a noun [39], whereas the "int" data type should probably be considered to be an adjective since it described the data being stored.

Each programming language possesses its own vocabulary and principles (or grammar). For this work we developed a set of features as listed in Table 3.1 from the Buse and Weimer work [9] and the C++ programming language. Some of the features and their meanings are described in appendix B. Below we present an overview of keywords, operators, loops, and comments.

- Keywords are identifiers that are predefined and have special meaning in a programming language [39]. These are words specified in the language definition which describe how data is manipulated and stored in memory. There are also some user-defined identifiers that are not part of the language definition and can be altered ac-

cording to the wish of the programmer.

- Operators are symbols specified in the definition of the C++ language to perform specific mathematical or logical operations. Operators are part of expressions, which are sequences of operators and operands that are used for the purposes of computing a value or designating a function.

- The execution of a group of statements (or block of code) multiple times or a certain number of times in a sequence is possible using various loops.

- Comments store additional information written in single or multiple lines within a program so that anyone can acquire an understanding regarding the line of code or functions. Their purpose is strictly to aid the person reading the program to understand what the program is supposed to do.

Table 3.1: List of 50 Features.

C++ Vocabulary	**Features**
Keywords	*#include*, *#define*, *using*, *void*, *cout*, *cerr*, *cin*, *return*, *exit*, *int*, *float*, *char*, *const* *double*, *bool*, *new*, *break*, *public*, *private*
Operators	$<$, $->$, $>$, &, &&, +, ++, !, ! =, $==$, =, $-$, $--$, $*$, $/$, $\|$, $\|\|$, /=, +=, -=, $*$=,$<$=, $>$=
Comments	//*, //, /* */
Brackets	{}, ( )
Block Execution	*for*, *while*, *switch*

To illustrate, consider the C++ program in Figure 3.1 which performs the summation of two numbers, ten times. In this program, we have:

- user-defined identifiers (or variables): *sum*, $firstNumber$, *secondNumber*, and i. All of the variables are of the same integer "*int*" datatype to store natural numbers $\{0, 1, \ldots\}$;
- various keywords: *#include*, *using namespace std*, *main*, for, *cout*, and *return*; and
- operators: =, $<$, ++, and +.

The keyword *for* in program Listing 3.1 (line number 8) represents the loop with counter "i" to specify the number of times each line of code within the loop (line number 9-12) will be executed. An expression $sum = firstNumber + secondNumber$; consists of two operators, the assignment (=) and addition (+) operators. The two expressions $firstNumber++$; and $secondNumber++$; contain the same increment (++) operator which will increase the value of each of the variables by one. The multi-line comment (/*) and the single-line comment (//) provide details about the program. Listing 3.2 contains the output of program Listing 3.1 which is the summation of two numbers performed 10 times.

Listing 3.1: Sample C++ Program.

```
1  # include "iostream"
2  using namespace std;
3  int main()
4  {
5  int sum;
6  int firstNumber = 1;
7  int secondNumber = 2;
8  /* This for loop runs the code for 10 times. */
9  for( int i = 0;i < 10;i++)
10 {
11 sum = firstNumber + secondNumber;          // addition of ↩
      two numbers
```

```
12 firstNumber++;                                  // add 1 in the ↩
      firstNumber
13 secondNumber++;                                 // add 1 in the ↩
      secondNumber
14 cout <<"Sum : "<< sum <<"\n";                  // print sum
15 }
16 return 0;
17 }
```

Listing 3.2: Output of the Program 3.1.

```
1 Sum = 3
2 Sum = 5
3 Sum = 7
4 Sum = 9
5 Sum = 11
6 Sum = 13
7 Sum = 15
8 Sum = 17
9 Sum = 19
10 Sum = 21
```

3.2 Step 1: Creating the Dataset

This research project was approved by the University of Lethbridge Human Subject Research Committee on March 5, 2012 under protocol number 2012-012. For this research, we collected C++ assignments and associated these with sociolinguistic information about the writer/programmer. The sociolinguistic information was assembled via a survey given

out to students in the computer science classes. Part of the survey included asking for permission to use the students' assignments in this research. The programming assignments were written by past and current students of the University of Lethbridge, as a requisite of their course work. Personally identifying information including names of participants, course numbers and titles, and names of instructors was sometimes included in the comments to the computer programs. This type of information was removed from the data before beginning our analysis to maintain privacy and confidentiality

In Table 3.2 we listed the classes, the number of participants, the number of individual samples, and the number of multiple samples from female participants excluding the individual ones. The classes of 1000 level (first year) represent introductory ones, 2000 (second year) demonstrate intermediary level, 3000 level (third year) are the advanced ones, and 4000 (fourth year) are the highest level even graduate students take it. The reason for having fewer multiple samples than single samples is that there are few female student and some female students provided us with multiple C++ assignments while other students gave us only one assignment.

Table 3.2: Data Collection.

Class Levels	**Female**	**Male**	**Samples**	**Multiple Samples (Female)**
1000	7	19	7	19
2000	1	8	1	2
3000	7	11	7	5
4000	4	12	4	5
Total	19	50	19	31

The dataset we gathered for this work consisted of a total of 100 C++ programs which are 50 male-written and female-written programs. In our dataset we have programs written to solve various problems arising in various domains of computer science. Some domains

are broad, such as image processing (which is addressed at the University of Lethbridge at a fourth year level) or computer graphics (which is addressed at a third year level). Others are more specific to a particular problem and we refer to them based on the name of the problem. For instance, the problem/domain that we refer to as the *Date* domain (which was a problem posed for one of the first year programming courses) is composed of two problems: one problem deals with the comparison of three date formats while the other problem calculates the number of days between two dates.

From the problems presented in the first year programming courses, we identified more domains based on the name of their problems, including Pig Latin (translate sentences from English into Pig Latin), anagrams of words (convert word to all of its possible anagrams), and examining temperatures (calculate averages, the lowest and the highest temperatures for a particular year). The domain of sorting strings (presented in the third year courses), analysis of DNA sequences to identify the aligned substrings from two sequences or strings (presented in the fourth year courses), just to name a few. In our dataset, sample programs are not evenly distributed on the basis of problems, domains, or levels. As shown in Table 3.2 for some of the problems or domains, there are no female-written programs. For some of the problems or domains there are no male-written programs. In some cases, one programmer might have developed multiple programs for different problems belonging to the same domain.

As indicated in Table 3.3, the data was male-skewed due to the low number of female students in the various computer courses. In order to balance out the dataset, it was necessary to oversample the submissions from the female programmers. Oversampling is the resampling of the samples (programs) that are associated with a particular class label in order to equalize the number of samples within the dataset [18]. We achieved this by utilizing either one or at most five different assignments written by the same female programmer, as reported in Table 3.2. For male students, we considered only one assignment. This way we were able to produce a balanced class dataset which had the same number of tuples/samples

for the class labels of male and female. The increase in the size of the dataset enabled the classification methods such as decision trees (J48), nearest neighbor (K^*), support vector machine (SVM), and naïve bayes (NB) to extract useful information about the male-written and female-written programs. As a result, the four developed models were able to accurately categorize the test dataset of computer programs on the basis of the programmer's gender.

Table 3.3: Information about the Dataset.

Gender	**Number of Participants**	**Provided Samples**	**Used Samples**
Male	65	240	50
Female	19	64	50
Total	84	304	100

The collected C++ assignments were treated as text documents in order to apply different machine learning algorithms in an attempt to distinguish the gender of the author/programmer. Each assignment was composed of one or multiple C++ (.cpp) files. In the case of assignments with various C++ files, we concatenated all of the files into a single file. In the dataset the maximum numbers of tokens and lines of code 2999 and 892 respectively. Each file was manually cleaned to avoid falsification of the identified features. The cleaning process involved the removal of white spaces and the removal of comments that contain information about the human subjects, courses, and instructors. Hence, there was no loss of important information from the data in terms of the frequency of various features, namely keywords, comments, operators, and brackets.

3.3 Step 2: Numerical Representation

In this research, the classification model took inputs as a vector of continuous (or numeric) feature values and gave outputs/results in the form of discrete (or nominal) values.

The class labels were nominal, that is, male and female. Using the list of features, the C++ assignments were transformed into the appropriate data format. The dataset was then used to train and to test all four classification models including decision trees, support vector machine, naïve bayes, and nearest neighbor.

Each computer program was treated as a text document. In each document the tf-idf value for an individual feature (as listed in Table 3.1) was calculated to convert the original dataset into a numerical representation. Feature occurrences over the entire dataset were represented using term frequency and inverse document frequency (tf-idf).

3.3.1 Term Frequency (tf) and Inverse Document Frequency (idf)

Each computer program is split up into tokens that are separated by white spaces. The term frequency (tf) factor is defined as the number of times a feature/term is found in a program [40]. For example, in a program (D_1) the frequency of occurrence of the feature "for" is 10 and the total number of tokens in program D_1 is 100. The "for" keyword represents a token through which we identified the occurrence of the "for" loops in a program. As a result, we computed the term frequency (tf) for the "for" feature in D_1 as 0.1.

To compute the inverse document frequency (idf) factor for a feature we applied the following formula:

$$\text{idf} = \log_{10} \frac{N}{n},$$

where N is the total number of programs that are present in the dataset and n is the number of programs in which a feature can be found. As an example, in a given dataset N is 100 due to the presence of a total of 100 documents. The "for" loops are present only in ten different programs, hence $n = 10$ for the feature "for". Thus, the idf factor for the "for" keyword, which represented the number of "for" loops in the entire dataset, is equal to 1.

In a program, for a given feature, the term frequency and inverse document frequency

(tf-idf) vector F is obtained by the multiplication of its two factors as demonstrated below:

$$\mathrm{F} = \mathrm{tf} * \mathrm{idf}.$$

After computing the F value for each feature over each sample, we recorded F values to produce a feature vector. We needed to normalize the F value of each feature to unit length [22, 40], in order to equalize the length of each feature vector. For this purpose, we used the normalization factor [40] as shown below:

$$\text{Feature} = \frac{F}{\sqrt{\left(\sum_{i=1}^{Occ} (F_i)^2\right)}},$$

where F represents the feature (or attribute) and Occ is the number of total features. For our research, the value of Occ is 50 as listed in Table 3.1.

3.4 Step 3 and 4: Machine Learning Methods with Testing Protocol

Our experiments used two techniques to handle data, in order to develop various classification models:

(i) the use of cross-validation technique, and

(ii) the use of the hold out method, that is, a fixed split of the dataset into two divisions.

The above techniques were reported in section 2.5. We performed six different experiments using 100 C++ programs as instances/tuples. However, we used a different number of features in several experiments:

- The first experiment consisted of 50 features (as listed in Table 3.1) and used the cross-validation technique.
- The second experiment consisted of 50 features and the hold out method.

- The third experiment used four features (as indicated in Table 3.4) and the cross-validation technique.
- The four experiment used four features and the hold out method.
- The fifth experiment used seven features (as listed in Table 3.5) and the hold out method.
- The last experiment used seven features and the cross-validation technique.

We expanded on each of the six experiments below. All six experiments used the same dataset and numerical representation (tf-idf) as reported above. In all of the experiments, data instances/tuples/samples and features/attributes were associated with the *male* and *female* class labels. To carry out experiments and to develop classification models we used WEKA software suite and SVM^{light}. We reported results from the six experiments in terms of precision, recall, f-measure, true positive (TP), false positive (FP), and true negative (TN). For each classification model the values TP, FP, and TN were the raw numbers of computer programs acquired from the confusion matrix.

The WEKA software suite (described in section 2.4) was used to extract information from the C++ dataset in order to construct three classification models. Using WEKA, various filters were used to produce the appropriate data file format (as described in section 2.4.2). The *NonSparseToSparse* filter was applied to convert the regular representation of the dataset into the sparse data format so that only nonzero features could be listed.

WEKA is one set of implementations of machine learning algorithms that is being used in this research. WEKA requires that the dataset to be formatted in a specific manner. This format is as follows: the user-defined name of the relation, the list of features and class labels, and data samples (as described in section 2.4). In every experiment, each instance consists of <attribute> <value> and the associated class label. The <attribute> is the index of the attribute which starts from zero and the <value> is the tf-idf quantity of each

attribute. The class label for programs that are authored by male programmers and represented as negative tuples/examples are denoted as "male". We use the "female" class label to represent positive tuples corresponding to programs written by female programmers.

The format for SVM^{light} requires data tuples with class labels and feature vectors to be described as <feature>:<value>. The <feature> is the index of the feature which starts from one. We used *-1* to denote the male-written programs which were referenced by negative tuples. To represent positive tuples denoting programs written by female students, we used *+1* as the class label. To categorize the same dataset (C++ assignments) in all of the experiments, we converted the sparse dataset into the appropriate format required for SVM^{light}.

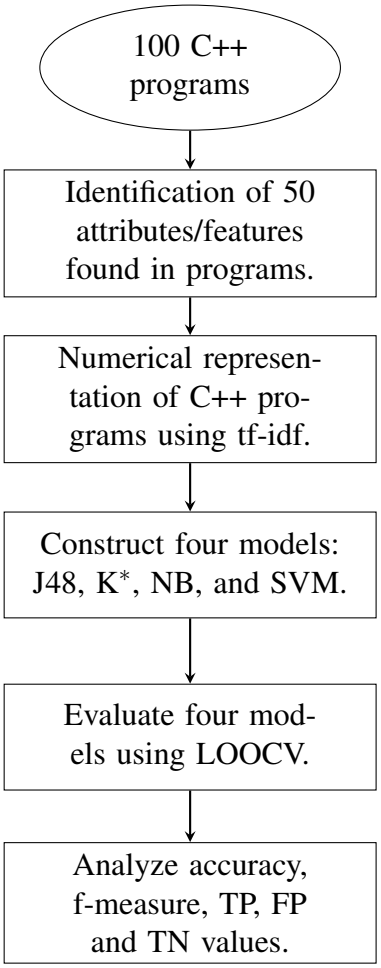


Figure 3.1: Experiment 1.

In the first (as shown in Figure 3.1) and the second experiment (as shown in Figure 3.2), we used a total of 100 tuples/samples and 50 features which were associated with the *male* and *female* class labels, as listed in Table 3.1. Using WEKA, three models were created including decision trees (J48), naïve bayes (NB), and nearest-neighbor (K*) models. The fourth model was developed using SVM^{light}. In the first experiment the leave-one-out cross validation (LOOCV) technique (described in section 2.5.2) was applied to evaluate each of the four models and the results were collected in terms of the evaluation metrics (described in section 2.5.3).

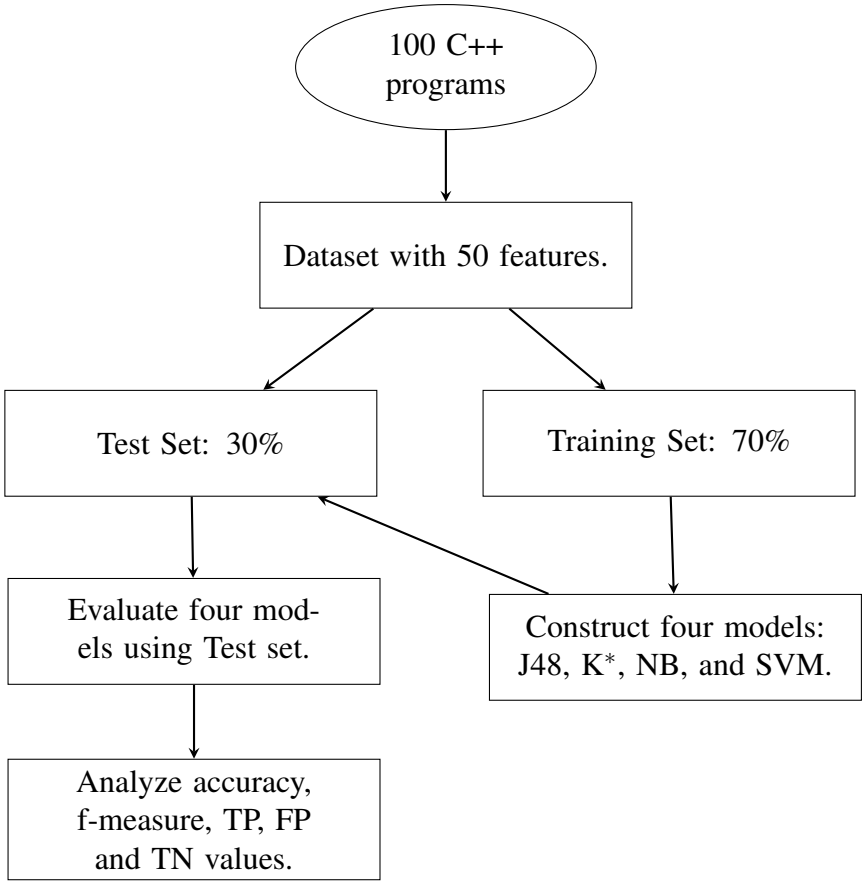


Figure 3.2: Experiment 2.

In the second experiment we applied the hold out method and separated the dataset into two partitions. In this experiment the learning step consisted of constructing the same four models using one partition of the data which was composed of 70% of computer programs. Next, in the classification step, we evaluated the predictive ability of the four models using the other partition with 30% of computer programs.

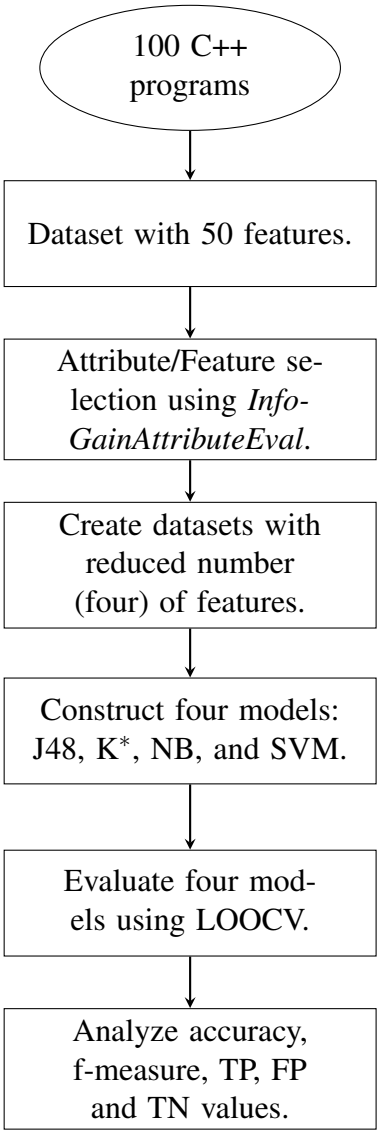


Figure 3.3: Experiment 3.

As shown in Figure 3.3 the third experiment used only four features (listed in Table 3.4). These features were selected from the dataset using the Information Gain statistical measure *(InfoGainAttributeEval)* for feature selection (described in section 2.4.7). This feature (or attribute) selection method used the *Ranker* algorithm, which ranked features, according to the Information Gain. After the application of information gain measure *(InfoGainAttributeEval)* and the *Ranker* method, we were able to find the highest ranked features among 50 features. This meant that, after the evaluation of every single feature, the *Ranker* sorted the features and produced a ranked list of features based on the information gain measure, as shown above in Table 3.4.

Table 3.4: List of Four Features.

C++ Vocabulary	**Features**
Keywords	*double*
Operators	/, ==, +

As a result, we found only four features that may have had an impact on the predictive ability of the classifiers after analyzing the values of the f-measure evaluation metric. We observed that the division operator achieved the highest ranking and the *double* datatype had the lowest ranking as listed below:

1. division operator (/),
2. equality operator (==),
3. addition operator (+), and
4. double datatype (*double*).

The *Remove* filter was utilized to remove the features with rank zero and to extract a dataset with a reduced number of features (the above four features only). This filter is part of WEKA and can be employed to exclude the remaining irrelevant features from the dataset.

The purpose of excluding the irrelevant features/attributes is to remove "noise" so that the classification models are not confused when decisions are formed. In the third experiment, the naïve bayes (NB), decision tree (J48), and nearest-neighbor (K^*) classification models were again created using WEKA. The transition of the dataset into the appropriate format, as mentioned above, was needed to construct the fourth model. The support vector machine classifier was trained for the dataset with a small number of features using the SVM^{light}. The leave-one-out cross-validation (LOOCV) was used to evaluate all of the developed models which mitigated the risk of learning from the small dataset and gathering inaccurate generalizations from the dataset.

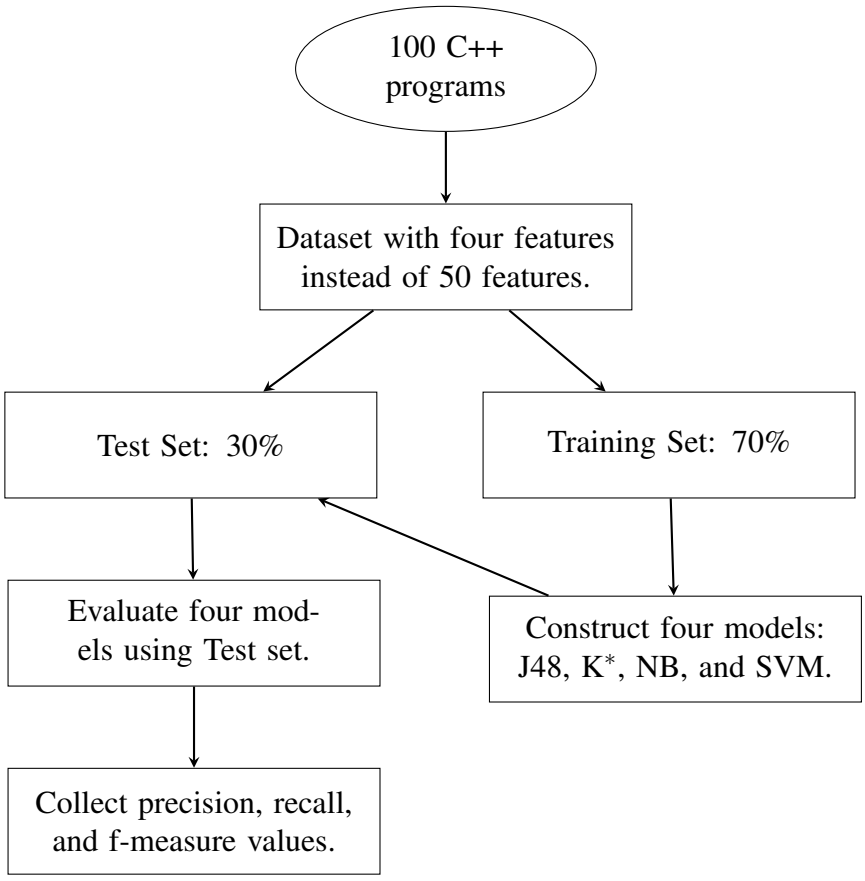


Figure 3.4: Experiment 4.

In the fourth experiment, we used the dataset created in the third experiment by using the information gain measure to identify useful features. We discovered only four features from the same 100 samples/tuples. The dataset was divided into two partitions using the hold out method (described in section 2.5.1). The overview of this experiment was demonstrated in Figure 3.3. One dataset contained 70% of the samples and was used to train each model. The other dataset was composed of 30% of the samples and was used to test each model.

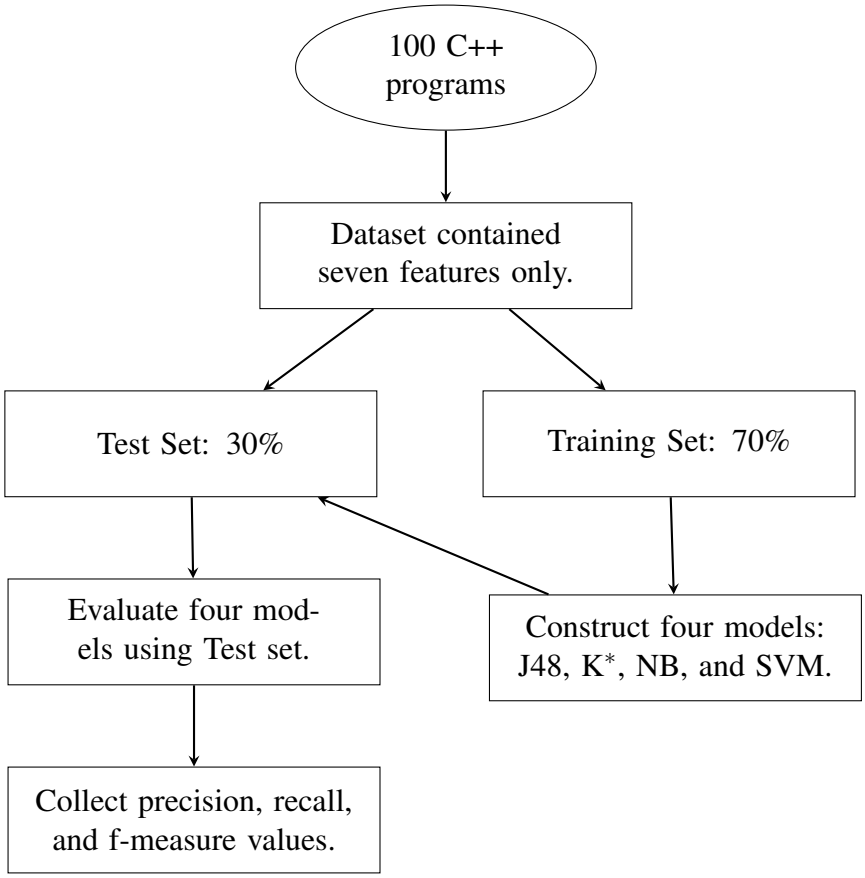


Figure 3.5: Experiment 5.

In the fifth experiment, as shown in Figure 3.5, we attempted to select a small set of features that played a role in identifying the gender of the authors of the C++ programs. We applied a correlation-based feature subset selection (*CfsSubsetEval*) evaluator [17]. This method worked with various algorithms, but we used a Genetic algorithm (as described in

section 2.4.7). As a result, we were able to apply both dimension reduction and feature extraction to our original dataset.

Table 3.5: List of Seven Features.

C++ Vocabulary	**Features**
Keywords	*double*, *char*, *bool*,
Operators	>=, +, ==, /

We found only seven features, as shown in Table 3.5. These were associated with the class labels but independent of each other. Among these seven there were the same four features (e.g. /, ==, +, *double*), which were identified by *InfoGainAttributeEval* and *Ranker* method as the highest ranked features. We applied the hold out method (described in section 2.5.1) to partition the dataset into two independent sets. The training partition again contained 70% of the programs while the test partition was composed of 30%.

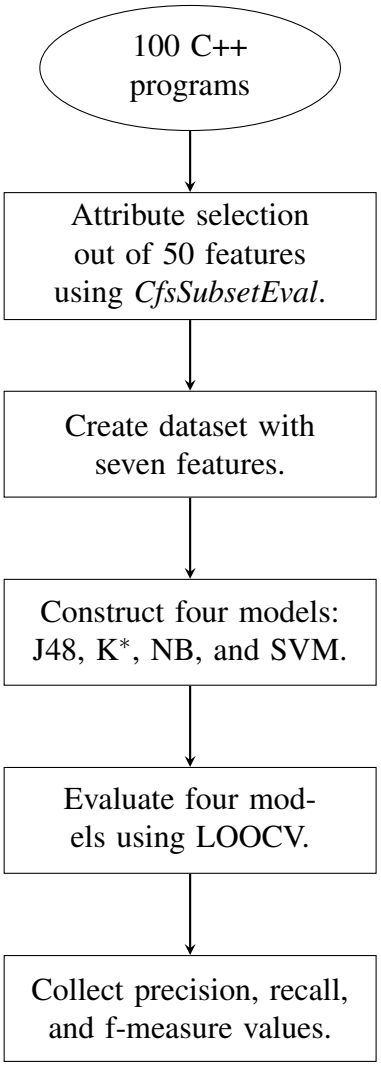


Figure 3.6: Experiment 6.

In the last experiment as presented in Figure 3.6 naïve bayes (NB), decision trees (J48), and the nearest-neighbor (K*) models, implemented as parts of WEKA, were created using only seven features, as listed in Table 3.5 above, using the same 100 instances. To develop the fourth model, the support vector machine, the dataset was converted into the required format using the approach described above in Experiment 1. The support vector machine classifier was built from the dataset with a small number of features using SVM^{light}. Leave-one-out cross-validation (LOOCV) was used to evaluate all of the developed models.

3.4.1 Parameter Settings

For all of the experiments which were reported above we tweaked a few parameters and left the rest with their default values [21]. Using WEKA machine learning software we employed supervised learning algorithms and attribute evaluators. We developed three out of the four models using WEKA; specifically, decision trees, nearest neighbor, and the bayesian classifier models. We also used attribute evaluators (information gain and correlation feature subset) to extract a set of reduced features from the original dataset. For the fourth supervised learning model (support vector machine model) we used SVM^{light} software.

- Using WEKA we applied the *Randomized* filter so that the instances in the dataset would be randomly shuffled before the application of the supervised learning methods [45]. To randomly shuffle the order of instances and the attribute values the seed value was set to the default value of 42. We set a seed value of 50 in all of the experiments because we had a small number of features (attributes) and instances (tuples).

- For the K* nearest neighbor algorithm, we changed the treatment mode for missing values (M) to "d" so that the model would ignore those values. The global blending (B) parameter value can be used from 0% to 100% [11]. We set the "B" to 70% to specify the number of neighbors to be considered significant.

- We set the *useKernelEstimator* (K) parameter for the naïve bayes algorithm to "true" so that the classifier did not assume any specific distribution of the dataset [45, 42].

- In the case of the support vector machine (SVM^{light} [21, 45]) we used a radial basis kernel function (t = 2) with a value of γ (g = 4.0). We applied a trade-off between learning error and margin using the (c) parameter with a value of 4.0.

- In the case of feature selection using *CfsSubsetEval* and *Genetic search*, we used the default settings [45]; however, we set the parameter *seed* of the Genetic method to the value of 8.

3.5 Results

In this section we discuss the results of six experiments to evaluate the predictive ability of the four models including decision tree (J48), nearest neighbor (K^*), support vector machine (SVM), and naïve bayes (NB). We use accuracy, precision, recall, f-measure, true positive (TP), false positive (FP), and true negative (TN) (see section 2.5.3). In each experiment the table shows the values of the three evaluation metrics, TP, FP, and TN. The description of these values are as follows:

(i) Accuracy is the ratio of computer programs that are correctly classified.

(ii) Precision is the ratio of computer programs that are correctly classified as female-written (positive tuples).

(iii) Recall gives the overall ratio of programs that are correctly and incorrectly classified as female-written (positive tuples).

(iv) F-measure, the common approach, is the harmonic mean of recall and precision.

(v) True positive (TP) is the number of programs that are classified as female-written programs.

(vi) False positive (FP) is the number of male-written programs that are classified as female-written programs.

(vii) True negative (TN) is the number of correctly classified male-written programs.

In the following sections we report results from the six experiments. In each experiment four classification models are developed with the goal of distinguishing between *male*-authored and *female*-authored computer programs. Our main interest lies in the models that achieve the highest values of the accuracy and f-measure evaluation metrics [9, 18]. A value greater than or equal to 70% for either of these metrics is used to further delve into the number of computer programs that are classified or misclassified by the models. This is done by exploring the values of TP, FP, and TN for each experiment.

3.5.1 Experiment 1

In this experiment, we used 50 features and did not scale down the dimension of our dataset. We used leave-one-out cross-validation (LOOCV) which performed the task of the division of a dataset into training and testing to return final results in terms of evaluation metrics. In this technique the entire dataset of 100 programs was used. As shown in Table 3.6, the SVM model performed well (accuracy = 75%). This model was able to correctly label the majority of the female-written programs (f-measure = 74.4%). In comparison with the other models, this model was able to categorize 36 out of 50 female-written programs (TP = 36/50) and 39 out of 50 male-written programs (TN = 39/50). This model misclassified a small number of male-written computer programs as female-written programs (FP = 11/30).

The K* model achieved the highest values in comparison with the J48 and the NB models. This model correctly labeled the majority of computer programs belonging to both classes (accuracy = 72%) and correctly classified the majority of female-written programs (f-measure = 71.9%). A total of 39 out of 50 female-written programs (TP = 39/50) and 33 out of 50 male-written programs (TN = 33/50) were correctly labeled by this model. However, this model misclassified 17 male-written computer programs as female-written programs (FP = 17/50) and we observed that the value of FP impacts the performance of this model in terms of f-measure.

Table 3.6: 50 Features and LOOCV.

Models	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F-measure (%)**	**TP (/50)**	**FP (/50)**	**TN (/50)**
J48	63	63	63	63	33	20	30
K*	72	72.3	72	71.9	39	17	33
SVM	75	76.6	72	74.2	36	11	39
NB	66	66	66	66	33	17	33

3.5.2 Experiment 2

In this experiment, we applied the hold out method to create a training partition and testing partition but used the same number of programs and features as in experiment 1. Using one partition (training set) of 70% computer programs we constructed the four classification models. The other partition (test set) of 30% computer programs was utilized to evaluate the performance of the models. The hold out method reduced the amount of data used in the learning and classification steps. Therefore, there was a risk that the models were either not able to learn accurate information or not able to correctly categorize a new dataset.

From Table 3.7 we investigated only those models which resulted in the accuracy and f-measure of more than 70%. The first model which performed well was SVM. The model correctly classified the computer programs from both classes and achieved an accuracy of 86.7% and an f-measure of 87.5%. Hence, in comparison with other models, this model was able to categorize 14 out of 15 female-written programs (TP = 14/15) and 12 out of 15 male-written programs (TN = 12/15). Only 3 male-written computer programs were misclassified as female-written programs (FP = 3/15).

The K^* and NB models achieved similar results. These models were able to accurately classify 73.3% of computer programs (accuracy = 73.3%). However, a difference occurred in terms of the f-measure. The K^* model resulted in an f-measure of 72.9% while the NB model resulted in an f-measure of 73.2%. The K^* model correctly labeled 13 female-written programs (TP = 13/15), mislabeled 6 female-written programs (FP = 6/15), and correctly labeled 9 male-written programs (TN = 9/15). The NB model correctly classified 12 female-written programs (TP = 12/15), incorrectly classified 5 female-written programs (FP = 5/15), and correctly classified 10 male-written programs (TN = 10/15).

Table 3.7: 50 Features and Test Set with 30% Samples.

Models	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F-measure (%)**	**TP (/15)**	**FP (/15)**	**TN (/15)**
J48	63.3	63.4	63.3	63.3	10	6	9
K*	73.3	75.1	73.3	72.9	13	6	9
SVM	86.7	82.4	93.3	87.5	14	3	12
NB	73.3	73.8	73.3	73.2	12	5	10

3.5.3 Experiment 3

In this experiment, the statistical measure of information gain was used to calculate the probabilities of the features on the basis of associated class labels. Features with an information gain value greater than zero were considered (as listed in Table 3.4). As shown in Table 3.8, the J48 model correctly classified the majority of computer programs as demonstrated by the accuracy measure. The fact that the J48 model achieved the accuracy of 70% interested us most. This model resulted in an f-measure of 69.4%. In comparison with other models, this model was able to categorize 42 out of 50 male-written programs (TN = 42/50). There were only 28 female-written programs labeled correctly by the model (TP = 28/50). This model misclassified a small number of male-written computer programs as female-written programs (FP = 8/50).

Table 3.8: Four Features and LOOCV.

Models	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F-measure (%)**	**TP (/50)**	**FP (/50)**	**TN (/50)**
J48	70	71.7	70	69.4	28	8	42
K*	69	69.1	69	69	33	14	36
SVM	67	65.5	72	68.6	36	19	31
NB	59	60.2	59	57.8	21	12	38

3.5.4 Experiment 4

For experiment 4, we used the same dataset created after the application of information gain measure. We applied the hold out method instead of the cross-validation technique and created two partitions. One partition was used to train the models, and the other partition was reserved for testing. In this experiment none of the models were able to achieve more than 70% accuracy or f-measure. However, we observed that the J48 model performs better than the remaining models as shown in Table 3.9. The model was able to correctly classify the majority of computer programs (accuracy = 66.7%). This model achieved the highest f-measure of 67.3%, classified 8 female-written programs correctly (TP = 8/15), misclassified 7 male-written programs (FP = 7/15) which had an impact on the f-measure, and correctly classified 12 male-written programs (TN = 12/15) which had an effect on the accuracy of the model.

Table 3.9: Four Features and Test Set with 30% Samples.

Models	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F-measure (%)**	**TP (/15)**	**FP (/15)**	**TN (/15)**
J48	66.7	70.2	66.7	67.3	8	7	12
K*	56.7	63.8	56.7	57	8	10	9
SVM	50	38.9	63.6	48.3	7	11	8
NB	56.7	61.3	56.7	57.4	7	9	10

3.5.5 Experiment 5

In this experiment, we extracted a subset of features using a correlation-based feature subset selection evaluator (*CfsSubsetEval*) with the genetic search algorithm. We discovered only seven features as listed in Table 3.5. The three models K*, SVM and J48 performed well with seven features in comparison with experiment 4 (four features). As listed in Table 3.10, the K* model achieved the highest accuracy and f-measure. The K* model resulted in an accuracy of 80% and f-measure of 79.9%, correctly classified 11 female-written programs (TP = 11/15), misclassified 2 female-written programs (FP = 2/15), and correctly classified 13 male-written programs (TN = 13/15).

The J48 model was able to label the programs with an accuracy of 73.3%. This model resulted in an f-measure of 73.2%, classified 10 female-written programs (TP = 10/15), and 12 male-written programs (TN = 12/15) correctly which had impact on the accuracy of the model. However, 3 male-written programs (FP = 3/15) were misclassified which was the same for the SVM model. The SVM model also performed well and correctly classified computer programs with accuracy of 70%. This model correctly labeled 9 female-written programs (f-measure = 66.7%, TP = 9/15). A total of 12 out of 15 male-written programs (TN = 12/15) were correctly classified.

Table 3.10: Seven Features and Test Set with 30% Samples.

Models	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F-measure (%)**	**TP (/15)**	**FP (/15)**	**TN (/15)**
J48	73.3	73.8	73.3	73.2	10	3	12
K*	80	80.5	80	79.9	11	2	13
SVM	70	75	60	66.7	9	3	12
NB	66.7	67.9	66.7	66.1	8	3	12

3.5.6 Experiment 6

In this experiment, we used a dataset with only seven features (as in experiment 5), 100 data tuples (samples), and employed the LOOCV technique instead of hold out in order to extract important information from the whole dataset. The K* and J48 models achieved similar results. In terms of accuracy these models were able to accurately classify 70% and 71% of the dataset. However, the difference occurred in the labeling of the female-written programs. The K* model resulted in an f-measure of 71% while the J48 model achieved an f-measure of 69.9%. The K* model correctly labeled 36 female-written programs (TP = 36/50) which impacted the f-measure, mislabeled 15 female-written programs (FP = 15/50), and correctly labeled 35 male-written programs (TN = 35/50). The J48 model correctly classified 32 female-written programs (TP = 32/50), incorrectly classified 12 female-written programs (FP = 12/50), and correctly classified 38 male-written programs (TN = 38/50) which had impact on the accuracy.

Table 3.11: Seven Features and LOOCV.

Models	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F-measure (%)**	**TP (/50)**	**FP (/50)**	**TN (/50)**
J48	70	70.3	70	69.9	32	12	38
K*	71	71	71	71	36	15	35
SVM	66	64.8	70	67.3	35	19	31
NB	61	62.4	61	59.8	22	11	39

3.6 Discussion

In this section we perform comparative analysis of the models on the basis of accuracy, f-measure, true positive (TP) and false positive (FP). Accuracy reflects the overall performance of the model in the recognition of computer programs that are male-written and female-written (both classes). We observe that in some cases there tends to be an inverse relationship between precision and recall due to the number of false positives [9]. For this reason, f-measure is used which combines the precision and recall into a single measure. The true positive (TP) and false positive (FP) measures demonstrate the performance of the model in terms of classifying female-written computer programs (that is, we are classifying the data into a single class).

The goal is to reach 100% f-measure and accuracy of a model [18, 9, 45]. Our main interest lies in values equal to or greater than 70%. For our dataset (with 50 samples) the goal is to achieve a true positive (TP) of 50 for the LOOCV approach and 15 for the hold out approach. In the case of LOOCV the entire dataset is used in the supervised learning and there are 50 programs associated with each class label. In the hold out method, models are evaluated using the test partition which is composed of 30% of the dataset. As a result there are only 15 programs for each class label. The best possible value for a false positive (FP) is 0 for all cases.

3.6.1 Method 1: Hold Out

The results from experiment 2 (section 3.5.2), 4 (section 3.5.4), and 5 (section 3.5.5) are discussed here to show the differences in the performance of the models. In these experiments the data division is performed using the hold out technique (see section 2.5.1). We use a fixed percentage of computer programs to divide the dataset into a partition with 70% and a partition with 30% of the programs. The first partition (70%) is used for training and developing (learning step) the model. The second partition (30%) is used for testing (classification step) the model. The experimental results are based on the performance of the models in the classification step of the supervised learning.

We notice that in experiments 2 and 4, SVM and K^* work well with 50 and seven features. SVM is the state-of-the-art supervised learning method. The NB model performs better only in experiment 2 (50 features). The performance of the NB model decreases in the presence of four features (experiment 4) and seven features (experiment 5) because there is not enough information in the reduced set to allow it to "learn".

The performance of the J48 model is improved in experiments 4 and 5. One of the reasons is that J48 is able to extract information from the dataset because of its feature selection method which identifies the most important features. The models from experiments 2, 4, and 5 are not considered for further analysis because of the hold out method. This method reduces the number of samples (by partitioning the entire data into two sets) for each class label (male and female). The models from the above experiments are not able to learn accurate information completely and might be unreliable to classify future datasets. Therefore, we applied n-fold cross validation (LOOCV) to develop and consider models for further analysis.

3.6.2 Method 2: Leave-One-Out Cross-Validation (LOOCV)

Unlike with the hold out method, the entire dataset is used in both training and validation for experiments 1 (section 3.5.1), 3 (section 3.5.3), and 6 (section 3.5.6). In each

experiment, there are a different number of features but the same four classification algorithms are implemented to develop the models. We use leave-one-out cross-validation (LOOCV). This is n-fold cross validation, where n=k and represents the total number of samples that are present in the dataset (see section 2.5.2). LOOCV automatically performs the two steps of the supervised learning. This method performs 100 iterations to train and test models because there are 100 computer programs in the dataset. The averages values of the evaluation measures from all iteration are used as the final performance estimates.

We observe in experiment 1 (using 50 features) that SVM outperforms K^*. This is likely because SVM extracts more generalized information from the dataset in high dimensional feature spaces [33]. Thus, the performance of both the nearest neighbor (K^*) model and support vector machine (SVM) model is better in comparison with other models. In experiment 3 we used a reduced set of four features and only J48 resulted in a better f-measure than the other models. In contrast, with the use of seven features (experiment 6) both K^* (71% of f-measure) and J48 (70% of f-measure) models performed well. These models are able to categorize the majority of computer programs accurately.

For further analysis of the feature usage in male-authored and female-authored programs we considered the subset of seven features in the male-authored and female-authored programs (see chapter 4). The reason for this is that results in experiment 6 (seven features) improve in comparison with experiment 3 (four features). The models do not perform quite as well as in experiment 1 (50 features) but still provide results that are very close to the 50 features results while offering a more manageable set of features for deeper analysis. In both hold out (method 1) and leave-one-out cross validation (method 2) the three models including SVM, K^*, and J48 are able to categorize computer programs better than the NB model. Overall we conclude that our experiments demonstrate that information about authors' gender can be identified from the computer programs written by these authors.

3.7 Threats To Validity

Buse and Weimer [9] observed that in this form of research some threats towards the validity of the results would be identifiable. We identify several potential threats to validity in our work:

(i) The dataset was composed of only 100 computer programs. The result of this threat can be observed in the fact that experiments with the hold out method achieved the highest results. On the application of the LOOCV method we were not able to acquire results as high, indicating that the small dataset gave insufficient data for training when partitioned further using the hold out method.

(ii) The number of female participants in our study. We had more male-written programs than female-written programs. For this reason, we resampled programs written by female programmers to increase the number of programs. Having programs belonging to various domains and solving varying problems could result in variation in feature usage that was not necessarily linked to the authors gender (or other sociolinguistic variables). Because we needed to resample to acquire an equal number of samples for each gender, we ran the risk of features connected to a particular domain/problem being identified as those used by female authors.

(iii) Another threat to validity is the level of courses from which various programs had been collected. The dataset was not evenly distributed across the class levels, that is, among first year, second year, third year, and fourth year. Most of the programs belonged to participants in the introductory and intermediate computer science classes at the University of Lethbridge. There is the possibility that these courses imposed a specific coding style regarding comments, indentations, and the use of data types.

(iv) In our experiments, the performance of the four classification models varies due to the different underlying mechanisms, application of two model evaluation techniques,

and different number of features. As we evaluate the model performance we need to be aware of these variations and their underlying (potential) reasons.

(v) We used a fixed and small set of features to classify male-written and female-written C++ programs. Our models were trained for these features. Thus, it may be possible that other important features from the C++ language were not part of our selected set of features.

3.8 Programming Environment

All experiments were run on a Mac Book Air, running Lion 10.7.2, with an Intel Core 2 Duo processor, and 2 GB of memory. The term frequency and inverse document frequency (tf-idf) of 50 features were calculated based on the formula given by [40] using the Java programming language. These features were manually identified and employed to represent 100 C++ samples. There were three tools used in the research: SVM^{light}, WEKA, and SPSS.

SVM^{light}, as described in section 2.3, is an open-source implementation of the support vector machine written in the C programming language by Joachims [23]. WEKA is also an open-source and is composed of various machine learning algorithms implemented by Witten and Frank [45, 20]. SPSS [3] is a commercial software package used to perform statistical analyses on datasets.

In this research we employed WEKA to implement various filters, three classification algorithms, and two attribute selection methods. Various filters were used to preprocess the dataset (as described in 2.4.2) including *NonSparseToSparse*, *StringToNominal*, and *Remove*. We implemented three supervised learning/classification algorithms (as reported in section 2.4.3) to categorize male-authored and female-authored C++ programs. We attempted to identify and select relevant features using two feature selection methods, as described in section 2.4.7. We used SPSS to investigate the correlation between features and gender (section 4.2). We also explored whether there exists a significance difference

between the use of features in male-authored and female-authored programs (section 4.2).

Chapter 4

Analysis of Features

In this thesis we began with the 50 features/attributes as listed in Table 3.1 in Chapter 3. As described in section 3.1, these features were selected based on the work in [9] and on the basic units of the C++ programming language. These features were chosen without considering how they might be associated with male-written and female-written programs. We found that the gender of programmers could be accurately predicted by the four machine-learning models with f-measures ranging from 63% to 74.22% (as reported in section 3.5.1).

4.1 Reducing the Set of Features

To delve further into the findings of the machine-learning models, we applied two feature evaluators to identify the most relevant features. The selected feature evaluators were information gain *(InfoGainfeatureEval)*, and correlation-based feature subset selection *(CfsSubsetEval)* (as described in section 2.4.7). The seven features listed in Table 4.1 were identified using *CfsSubsetEval*. A subset of these $/$, $==$, + and *double* were also the highest ranked features as found by the *InfoGainfeatureEval*. These features were then used to develop the same four classification models using the leave-one-out cross-validation (LOOCV) technique to identify whether a subset of features was enough to categorize computer programs as either male-written or female-written programs based on the gender of programmers.

Table 4.1: Subset of Features.

Features	**Representation**	**Meaning**
Double	*double*	Store a 64-bit floating point value.
Boolean	*bool*	Store one of two values (True or False).
Division	/	Perform the division operation.
Addition	+	Perform the addition operation.
Character	*char*	Store a character value (e.g. 'c').
Equality	==	Compare values to check if they are equal.
Greater than or equal to	>=	Check if a value is either greater than or equal to a second value.

Table 4.2 shows the performance of the four models with 50 features, four features, and seven features. Using only four features the J48 model achieved the highest f-measure in comparison with the remaining models. However, when we broaden the analysis to include all seven features found by *CfsSubsetEval*, we see that K* results in the highest f-measure in comparison with the other models. The set of four features reduces the performance of models, but with seven features we can still obtain a 71% f-measure.

Table 4.2: F-measures Based on LOOCV.

Classification Algorithms	**F-measure Based On 50 Features (Experiment 1)**	**F-measure Based On Four Features (Experiment 2)**	**F-measure Based On Seven Features (Experiment 5)**
J48	63%	**69.4**%	69.9 %
K*	71.9%	69%	**71**%
SVM	**74.22**%	68.56%	67.30 %
NB	66%	57.8%	59.8 %

We performed comparative analyses for the experiments using the reduced set of features and the LOOCV technique, as shown in Table 4.2. There is an improvement in the performance of the J48 model for the dataset with the seven features; however, this is not the case for the remaining three models. One reason for this improvement is the underlying concept of the J48 classification algorithm. This is a basic decision tree, constructed recursively by partitioning tuples into associated classes. During the construction of a tree the algorithm applies an attribute selection evaluator known as the gain ratio, to determine the "best" attribute [38, 17]. This way the attribute is selected that extracts the maximum amount of information from the dataset to distribute tuples into distinct classes.

The SVM model performs the worst with the seven features in comparison with 50 features because the number of features plays a role in finding the "best" maximum margin hyperplane [21, 33]. The J48, K*, and NB models result in higher f-measures with seven features than with four features as discussed in section 3.6.2. One possible reason for the increase is that reducing to four features removes too much information. The addition of a few features back to the subset of four features increases performance, although not quite as high as with 50 features. Is the difference significant? It may not be, but the results are close, as discussed below.

To further explore only models with the highest f-measures on the basis of subsets of

four and seven features from experiments 2 and 5 (described in section 3.5.2 and 3.5.5) we used a confusion matrix (described in section 2.5). The computation of f-measure is based on the data from the confusion matrix (as described in section 2.5.3). In experiment 2, the f-measure of 69.4% is achieved by the J48 model which is better than the rest of the models. This model accurately classified 28 out of 50 as female-written programs and 42 out of 50 programs as male-written programs. As shown in Table 4.3, 70 programs were correctly classified; however, there were 22 programs mislabeled as male-written and 8 programs mislabeled as female-written. One of the drawbacks of the J48 model is the overfitting of data; we noticed that the model is more biased towards the *male* class label. The model is not able to categorize female-written programs accurately.

Table 4.3: J48 Confusion Matrix (Experiment 2, with four features and LOOCV).

GENDER	F	M	TOTAL
F	28	22	50
M	8	42	50
TOTAL	36	64	100

In experiment 5 the K* model achieved the highest f-measure of 71%. Table 4.4 shows that the K* model accurately predicted the gender of 36 female authors and 35 as male authors; thus, 71 programs were correctly classified. We noticed that the model is not biased towards one class label. The K* model incorrectly predicted gender for 14 female-written and 15 male-written programs.

Table 4.4: K* Confusion Matrix (Experiment 5, with seven features and LOOCV).

GENDER	F	M	TOTAL
F	36	14	50
M	15	35	50
TOTAL	51	49	100

Selecting a subset of features allows us to perform more detailed analyses to explore gender differences, which we will discuss in the remainder of this chapter. First, we analyze the frequency of the seven features within the dataset. Secondly, we briefly discuss the dataset in term of programs and their problem domains. Thirdly, we carry out visual analysis of the shortest and the longest male-written and female-written programs. Fourthly, we perform problem-specific analysis. Finally we explore the correlation (γ) of the seven features within the male-written and female-written programs.

4.2 Statistical Approach

In this thesis we are dealing with two groups of programmers which are "male" and "female". We are interested in examining how programmers in these groups develop computer programs using various programming styles which are related to the use of different features. We hope to determine whether the differences are real or occur by chance. A statistical test is performed to determine whether observations are statistically significant or not in terms of the ρ-value that can be found in the output.

The statistical technique *t-test* is a "bivariate statistical test" [12] and is a common approach to compare two groups in terms of their means. We carried out a two-tailed test to find out whether either mean is greater than the other one. In other words, we are interested in determining whether the differences are greater than a random chance in order to determine whether the difference can be considered to be statistically significant. In t-test a null hypothesis is tested against the alternate hypothesis. The null hypothesis (H_o) states

that there is no difference in two groups ($\mu_1 = \mu_1$) because it might occur by chance. The alternate hypothesis (H_a) stated that there is a difference in two groups ($\mu_1 \neq \mu_1$). The result from the t-test either supports or rejects the H_o. If the null hypothesis is rejected then the alternate hypothesis is supported but cannot be proved. The ρ-value provides evidence that either the difference is due to the random chance or not [12]. A threshold value of 0.05 is used to determine the statistical significance of the difference between two groups. The ρ-value less than 0.05 represents that the difference is real which leads to the rejection of the null hypothesis (H_o). The ρ-value that is greater than 0.05 demonstrates that the likelihood of the result occurring by chance is high and H_o (the null hypothesis) can be accepted. Thus, the results are non-significant.

One of the reason that this might occur is the absence of a large number of samples due to which it is not possible to find a statistically significant difference. Studies with a small number of samples are bound to have "deficient power" [12], which means that it is not possible to know if results would be different in the presence of a large number of samples. In the following sections we analyze the use of seven features and visually identified features to determine if the difference in the use of features between the two groups of programmers is statistically significant.

4.3 Frequency of Occurrence

We first investigated which of the seven features are more likely to appear in male-written or female-written programs. Argamon et al. [5] identified gender-based differences in the usage of features by computing the feature frequencies per 10,000 tokens or words. Their results showed that the frequency of determiners was higher in male-authored documents while the frequency of pronouns was higher in female-authored documents. In this section we follow a similar approach to evaluate the reduced set of seven features in terms of their frequency of occurrences.

We use two different methods to compute the frequencies. In the first method we use

raw frequencies (u) of each feature within our dataset by ignoring the total number of tokens (v) that are part of a computer program. In other words, we count occurrences of each feature within a program without considering how many tokens are present in that program (that is, the length of the program). In the second method we divide the frequency (u) of each feature within a program by the total number of tokens (v) and multiply the result by 100, that is, $(\frac{u}{v} * 100)$. We use the second approach in order to determine whether the length of the program has an impact on the frequency of features.

Table 4.5: Means of Frequency Per 100 Tokens.

Features	**Female**	**Male**
/	0.294	0.379
==	0.789	1.322
+	0.924	0.815
>=	0.077	0.154
double	1.036	0.152
char	0.543	0.315
bool	0.361	0.378

Table 4.5 shows that the mean frequency of each feature within the dataset is smaller in the token-based calculation. The mean values for the features /, ==, >= and *bool* are higher for male-written programs. The mean values for the features +, *double* and *char* are higher in female-written programs. These values are extracted from C.3 and C.4. For each feature we are interested in determining whether the occurrences of these seven features are different in male-written programs and female-written programs.

Table 4.6: Seven Features Frequency Means and Standard Deviations.

Features	**Female** μ_F (σ_F)	**Male** μ_M (σ_M)	ρ-value
/	1.8 (2.19)	3.36 (4.75)	.039*
==	7.04 (12.67)	10.88 (12.7)	.133
+	6.38 (10.16)	6.3 (7.28)	.964
>=	0.44 (0.95)	1.04 (1.68)	.032*
double	4.82 (9.55)	1.22 (4.59)	.019*
char	2.58 (3.67)	1.78 (2.15)	.187
bool	2.18 (2.68)	2.82 (3.19)	.281

Table 4.6 shows the mean (μ), standard deviation (σ), and the test of significance (ρ-value) of frequency of each feature within the dataset.μ and σ are calculated to demonstrate the variation in the occurrences of features in the female-written programs (Table C.1) and male-written programs (Table C.2). To further explore the significance of differences among the frequency of the seven features in male-written and female-written programs we carried out the t-test using SPSS [3]. We use the t-test to determine which hypothesis is acceptable for the each feature (dependent variable) in the group (independent variable) of programmers. We have a total of 100 programs in our dataset. There are 50 programs written by both female and male programmers. In the case of female programmers we have

multiple programs from the same author; however, the change in the use of features in one program does not impact on the other programs. Thus, we can say that each program is independent in both groups [12].

As described in section 4.2, we observed that in Table 4.6 the statistically significant differences (values with $*$) are found in the use of three features: $/$, $>=$, and $double$. Therefore, for these three features the alternate hypothesis (H_a) cannot be rejected and we consider the findings to be significant. However, for the remaining features the null hypothesis (H_o) cannot be rejected and the findings are not considered to be statistically significant. In other words, the use of the $/$, $>=$, and $double$ features were found to have statistically significant different frequency usage when comparing male-written and female-written programs.

We also considered that possibly visual inspection might identify differences that the machine learning or frequency analysis did not identify, and so in the following section we investigate this possibility.

4.4 Visual Analysis of the Shortest and the Longest Programs

In this section we visually investigate the seven features by examining the shortest and the longest male-written and female-written programs. We found that the shortest program written by a female had 141 tokens. In this program we found only the + operator out of the set of seven features. Additionally, comments and print statements (*cout*) occurred more frequently in the shortest female-written program than in the shortest male-written program.

The smallest male-written program consisted of 107 tokens. Among the seven features only the $==$ operator was found in the program. There was one *cout* statement and there were no comments. The shortest male-written and female-written programs belonged to different domains of computer science. The female program was written as a computer game of rock-paper-scissors while the male program was written to calculate benefits of

timely bill payment. Of the seven features, we observed that the == operator appeared more in the male-written program and the + operator appeared more in the female-written program. However, in addition to the seven features we identified that there was a difference in the use of other features that were not in our subset, notably comments (as described in Table B.3) and print (*cout*) statements, which were used much more by the female author.

The largest male-written program had a total number of 2999 tokens while the largest female-written program had 2950 tokens. The longest male-written and female-written program both belonged to the domain of computer graphics. They were written to solve different computational problems. The female-written program developed to draw multiple pentahedra. The male-written program constructed multiple hypnocubes and colored tetrahedron. The /, +, and >= features appeared more frequently in the male-written than in the female-written program. In the female-written program ==, *char*, *bool*, comments and print (*cout*) statements appeared more frequently than in the male-written program. The differences in the use of comments and print statements within the largest male-written and female-written programs appear to be a factor in the gender-based differences for the number of tokens.

To further explore the gender differences around the use of comments and print statements we counted their raw frequency in each male-written and female-written program. In male-written programs the total frequency of comments was 1274. The total frequency of print statements was 461. In female-written programs the total occurrence of each of these was lower, 1219 and 344 respectively. These findings contradict the results from the shortest and the longest programs which found that comments and print (*cout*) statements appeared more frequently in the female-written programs than in the male-written programs. Therefore, to determine whether the observations are statistically significant or not we carried out t-test and used cutoff ρ-value of 0.05. As shown in Table 4.7 all ρ-values are greater than 0.05; hence, there are no significant differences in the use of *cout* and comments features between the two groups of programmers.

Table 4.7: T-test for Two Features

Features	**Female** μ (σ)	**Male** μ (σ)	ρ-value
cout	25.48 (26.14)	9.22 (11.97)	.259
comments	24.38 (33.4)	6.88 (8.27)	.855

After investigating the gender-based differences for these features we also observed that the programs in the dataset were developed to solve various computational problems either from the same or the different domains of computer science. In the next section we examine the above findings regarding the seven features, comments, and print statements using programs that solve the same computational problem in a specific domain.

4.5 Problem-specific Analysis

Could the differences in the use of the features depend on the type of the problem? Or do variations occur based on the domain of the problem? To investigate the above questions, we chose the following eight programs that each addressed the same problem. Four samples were chosen from each of male and female authors. This gave us a small dataset with consistency in both the domain and the problem for this analysis. The programs were all written to solve a problem in which each program takes three dates in different formats as the input, and prints the relationship between these dates. One step the program often made was to check whether the dates were the same or not.

Table 4.8: Domain Specific Female-written Programs

/	==	+	>=	*double*	*char*	*bool*	*cout*	//	//*	/* */	**Tokens**
4	42	6	1	0	0	6	32	41	0	0	956
4	16	6	0	0	0	8	16	7	2	10	1039
4	12	7	0	0	0	6	18	11	0	1	805
4	55	6	3	0	0	9	19	0	0	1	901
16	**125**	**25**	**4**	0	0	**29**	**85**	**59**	**2**	**12**	3701

Table 4.8 shows the number of occurrences for each feature in each of the female-written programs. The sum for each feature over all four programs is given in bold in the last row of Table 4.8. This table shows that ==, >=, and multi-line (//* and /* */) comments have higher occurrences in the four selected female-written programs. In these programs, == and >= are frequently used in conditional statements to check if two values are equal or not in order to make a decision. The >= feature is used to compare numerical values. The == operator is used to compare various kinds of values. Comments offered additional information to the human reader, potentially increasing the readability of the program.

Table 4.9: Domain Specific Male-written Programs

/	==	+	>=	*double*	*char*	*bool*	*cout*	//	//*	/* */	**Tokens**
4	43	6	0	0	1	6	30	0	0	1	854
4	18	6	0	0	0	7	31	9	0	0	879
4	17	7	0	0	1	6	29	70	0	0	1077
5	24	6	0	0	0	12	15	70	0	0	1088
17	**102**	**25**	0	0	**2**	**31**	**105**	**149**	0	**1**	3898

Similarly, Table 4.9 shows the number of occurrences for each feature in each of the

male-written programs. The sum for each feature over all four programs is given in bold in the last row of Table 4.9. This table shows that the frequency of the *char*, *cout*, and single-line comments (//) is higher in the four selected male-written programs than in female-written programs. The *char* feature is used to define the data type of variables in the programs. The *cout* feature is used to print text to the computer screen which relays information to the user of the program. A comparison of Tables 4.8 and 4.9 shows that the frequency of the + operator is the same, and the *double* data type does not appear in either male-written or female-written program samples. The features ==, >=, and multi-line (//* and /* */) comments have occurred more often in the four female-written programs than in the four male-written programs.

The gender-based differences in the use of seven features could depend on the specific problem domain. Although == appears more in the longest female-written program and >= appears more in the longest male-written program, we see that in female-written programs belonging to the three dates problem both of these features appear more frequently than in male-written programs. However, the longest programs (both female and male authored) were written to solve problems in the domain of computer graphics. Furthermore, in the 8 programs from the three dates problem females write programs with more multiline comments than males, while males write programs with more *cout* statements than females.

We again used the t-test to determine whether the difference in the use of features in the Date domain are statistically significant. We observe in Table 4.10 that ρ-values for all features are greater than 0.05 indicating that the results are not statistically significant. This is not surprising, given the small number of programs in the Date domain. However another possible reason for this is that it is possible that the use of features is not related to the problem domain.

Table 4.10: T-Test (ρ-values)

/	==	+	>=	*double*	*char*	*bool*	*cout*	//	//*	/* */
.391	.652	1.00	.252	-	.182	.771	.377	.341	.391	.326

Additionally, we were interested in analyzing connections between pairs of the seven features. Although *CfsSubsetEval* already offers some of this information, we analyzed how the pairs were connected within our dataset. In the next section we use *Pearson's Product Moment Coefficient* (γ) [18] and two-tailed test of significance in order to identify which pairs of features, within the subset of seven features, are strongly dependent on each other.

4.6 Relationship between Features

In this section we investigate whether there is a relationship between pairs of features. Our interest lies in identifying which pairs of features are linearly dependent on each other within the male-written and female-written programs. Using γ we cannot infer a cause-effect relationship, but we can characterize the programming style of males and females by examining the strength of relationship between features. We used Statistical Package for the Social Sciences (SPSS) [3] to compute γ and ρ-value for pairs of features across male-written and female-written programs within the entire dataset. The value of γ is useful to demonstrate the existence of a linear relationship between two features within the dataset of programs. Based on the value of γ there are three possibilities [18]:

(i) $\gamma = 0$ shows that features are independent and they are not correlated with each other;

(ii) $\gamma > 0$ shows that features are positively correlated so a higher value of γ shows that there is a strong correlation between the given features; and

(iii) $\gamma < 0$ shows that features are negatively correlated. This means that as the occurrence of one feature increases, the occurrence of the other feature decreases.

In Tables C.5 and C.6, we list γ values in a correlation matrix on the basis of the original feature frequencies. We evaluate the value of γ for each pair of features to determine whether there is no relationship, a positive linear relationship, or a negative linear relationship. A correlation (γ) of greater than 0.5 indicates a strong linear relationship [9]. In the following we examine each pair with γ value greater than 0.5 and ρ-value less than 0.05, as shown in Tables 4.11 and 4.12.

Table 4.11: Strongly Correlated Pair(s) Based on Raw Frequency in Male-Authored Programs.

Feature 1	Feature 2	γ	ρ-value
/	+	0.817	.000*
==	*bool*	0.73	.000*

Table 4.12: Strongly Correlated Pair(s) Based on Raw Frequency in Female-Authored Programs.

Feature 1	Feature 2	γ	ρ-value
/	+	0.356	.011*
==	*bool*	0.535	.000*

4.6.1 Pair 1: operators "/" and "+"

In male-authored programs we see that the pair / and + has a stronger positive linear relationship as demonstrated by γ = 0.817 and ρ-value = .000 which represents the significance as shown in Table 4.11. For both male-authored and female-authored programs there is a strong positive linear relationship between / and +, and it is statistically significant for both. However for male-authored program the correlation is stronger, as listed in Table 4.12.

In both groups these operators are used to perform different kinds of arithmetical opera-

tions and work with numerical values. The operator / is used to perform division while the + operator is used to perform addition. One of the reasons for the difference in the strength of the relationship could be that in our dataset we have multiple problems within the domain that are solved by the same female programmer. This is not the case in the group of male programmers.

To further explore how these features correlate with gender we examine differences based on programs in which the pair of features appear together. We observe that in the dataset / and + operators appear frequently in male-written samples that belong to the domains of computer graphics (third year), image processing (fourth year), binary search (second year), and dates (first year). The two features in female-written programs belong to the domains of computer graphics, dates, circle, and DNA analysis. We find that one of the reasons is that in the dataset there are more male-written samples in domains which solve a specific problem such as the three dates problem, hypnocube problem, binary search problem, and the problem of image processing. In our dataset, the problems of circle and DNA analysis have more female-written samples. The three dates problem and hypnocube problem both have male-written and female-written participants. In the following discussion we explore differences in the use of / and + operators on the basis of computational problems and their domains.

4.6.2 Male-authored Programs

The pair / and + has a very strong linear relationship in male-written programs. These operators appear together in 37 male-written (out of 50) programs, as shown in Table C.2. Each program is written by an individual programmer because we have more male participants and so it was not necessary to use multiple samples from a single author. In these 37 programs there is one program in the domain which solves the problem of creating different text boxes. In the domain of computer graphics there are four programs written to deal with the problem of simulating multiple hypnocubes and colored tetrahedron. There

are eight programs written to handle lists for binary search. There are 17 programs which work with dates and there are 8 programs in the domain of image processing. Thus, / and + operators appear in the male-authored programs belonging to the domains of computer graphics, binary searches, image processing, and dates.

4.6.3 Female-authored Programs

In female-authored programs the pair of features / and + has a weaker linear relationship than in male-authored programs (see Tables 4.11 and 4.12). In the dataset there are 28 (out of 50) programs that use both operators, as shown in Table C.1. In the group of female programmers we have multiple samples that belong to the same participant because of the underrepresentation of female participants in the field of computer science. There are 13 unique female authors. There is one program in each domain in which this pair appears including image processing; sorting strings; linked lists; and anagrams. There are twelve programs written to solve different problems in the domains of computer graphics and dates. There are six programs belonging to each of the above domains.

In the domain of computer graphics one problem deals with the simple simulation of hypnocube, and the other problem deals with multiple hypnocubes and colored tetrahedron. There are a total of five programs belonging to the three dates problem and one program solves the two dates problem in the date domain. There are two programs that belong to the domain of examining temperatures. There are three programs that calculate the area of a circle passing through multiple points. There are four programs that belong to the domain of bioinformatics in which two different problems have been solved: one is to find the local alignment of the given DNA sequence and the other is to find the global alignment. We see that the pair of features / and + appears in the female-authored programs that belong to the domains of computer graphics, dates, circle, and DNA analysis.

4.6.4 Pair 2: data type "*bool*" and operator "=="

In male-authored samples the pair of *bool* and == has a strong positive linear relationship (Table 4.11); however, in female-authored samples this pair has a moderate linear relationship (Table 4.12). For both groups the ρ-value is .000 which indicated that the results are statistically significant. In the domain from first year including anagrams, Pig Latin, and three dates the *bool* is frequently used as the method return type and very rarely used as a data type for a variable. The == operator is used to check equality of string ("yes"), numeric and bool (True/False) values. In our dataset we find that this pair occurs in male-written samples that belong to the domains such as binary searches, dates, and computer graphics. In female-written samples this pair appears in programs belonging to the domains including anagrams (first year), computer graphics (third year), and dates (first year).

One reason for this could be that in the dataset there are more male-written samples in domains which solve a specific problem such as the three dates problem, the hypnocube problem, and the binary searches problem. Similarly, the female-written programs are more in the domains which solve the problem of anagrams and pentahedra. The problem of three dates problem and the hypnocube problem both have male-written and female-written participants. Next, we will discuss gender differences in the occurrences of the *bool* and == features on the basis of computational problems and their domains in order to see how they vary in each group of programmers.

4.6.5 Male-authored Programs

In male-authored programs the == and *bool* features have a much stronger linear relationship than in female-authored programs. Table C.2 shows that the two features appear together in 31 programs (out of 50). These programs encompass several different domains of computer science. 17 male-written programs belong to the domain of dates. Eight programs handle lists for binary searches. There are 5 programs that belong to the domain

of computer graphics and solve the problem of creating multiple hypnocubes and colored tetrahedron (same problem as females). There is one male-written program that deals with the domain of converting words from English to Pig Latin (same domain as females). Thus, the pair of features *bool* and == is most frequent in the male-authored programs belonging to the domains of computer graphics, binary searches, and dates.

4.6.6 Female-authored Programs

Examining Table C.1 shows that the two features *bool* and == appear together in 23 (out of 50) female-authored programs. In the domain of dates, five programs solve the three dates problem. Six programs solve the anagrams problem. There are two programs that belong to the domain of translating words from English into Pig Latin. In the female-written programs there are three problems in the domain of data structures: linked lists, maps, and multisets. For each problem there is only one program which is written by the same programmer. There is one program in the domain of compilers. In the domain of computer graphics there are two programs related to multiple hypnocubes and colored tetrahedron while one program deals with the simple simulation of hypnocube. We also find that there are two programs that solve the new problem of simulating multiple pentahedron with different styles belonging to the domain of computer graphics. Hence, the two features appear more frequently in the female-authored programs belonging to the domains of computer graphics, date, and anagrams.

As a result of analyzing our dataset on the basis of ρ-values we find that the pair of == and *bool* has a significant positive linear relationship in female-authored programs and in male-authored programs, as shown in Tables 4.11 and 4.12. However, in terms of γ this pair is more positively correlated (linearly dependent on each other) in males than in females. Another gender-based difference occurs in the pair / and +. On the basis of γ this pair has a stronger positive linear relationship in male-authored programs than in female-authored programs. This pair has a weak correlation in female-authored programs.

After analyzing the dataset on the basis of correlation we are aware that the dataset is not evenly partitioned in terms of domains of computer science. Various computational problems are either from the same or different domains. Some problems in specific domains have not been solved by either female or male programmers, and some problems have a greater number of male participants than female participants. On the basis of correlation we cannot make unwarranted claims about the pairs of features because this measure does not infer a cause-effect relationship. Thus, γ only measures the strength of the linear relationship between the features and the ρ-value provides sufficient evidence towards the significance of the above findings.

Chapter 5

Conclusion

Sociolinguist William Labov [31] demonstrated that social variability may influence linguistic variability. In the society, the social variability might be seen in socio-economic status (SES), age, ethnicity, region, and gender on the basis of language use [27, 5]. Similarly, within the society of programmers there may be a possibility of finding variations in the use of programming language due to social factors. We selected gender as the single social variable to determine differences in the use of language. Hence, in this work we have investigated whether we can determine if gender, as a social factor, influences the development of computer programs written in the C++ language.

This study is based on the work of Argamon et al. [4, 5, 27], which categorized French and English text documents on the basis of the author's gender. They used various supervised learning and statistical techniques to analyze the language usage in male-authored and female-authored text documents. Thus, if gender-based categorization can be carried out via supervised learning in two natural languages, then this opens the door to extrapolation to artificial languages. For this reason, the aim of this study is to investigate the effects that gender might have on the use of programming languages.

Programming languages provide much less room for linguistic variation than natural language. The syntax of a program is quite strictly determined by the programming language. The choices left up to the programmer include the approach to solving the computational problems, spacing and the use of keywords, operators, statements, and comments. When put this way the breadth of choice for a programmer seems quite large. Misek-Falkoff

[34] suggests that techniques from linguistics can be used to analyze the language used to develop software, which is composed of computer programs. This gives additional support to our proposal to use natural language techniques in an artificial setting.

In this work we used various techniques from the area of automatic text retrieval, machine learning, and statistics. First, we used tf-idf technique to create a numerical representation of computer programs (as described in section 3.3). This technique is widely used in the area of automatic text retrieval [40]. We used a dataset of 100 C++ programs which were written by male and female programmers. The underrepresentation of females in the field of computer science prevented the collection of enough sample programs from female participants (as shown in Table 3.3). For this reason, we have a small number of samples overall for some particular problems. Also, for some problems there are not enough samples from both genders.

Secondly, we used various machine learning methods to categorize male-written and female-written computer programs. These methods included support vector machines (SVM), decision trees (J48), naïve bayes (NB), and nearest-neighbor (K*) (as described in Chapter 2). These methods have been used to categorize text documents in various natural languages [4, 33, 22]. As far as we are aware, this is the first attempt to pursue the idea of categorizing C++ programs on the basis of the author's gender, and to investigate gender differences on the basis of language use.

In machine learning various algorithms can be used to develop classification models. We performed five experiments using different algorithms (as described in sections 2.2.1 and 2.4.3), different ways of partitioning the dataset into training and test data (as explained in section 2.5), and different numbers of features using various feature selection algorithms (as described in section 2.4.7). In the first experiment we used 50 features and applied the leave-one-out cross validation (LOOCV) technique (as described in section 3.5.3). In the best case scenario we were able to correctly categorize 74.2% of the programs based on the gender of the author. In the second experiment we used 50 features and applied hold out

methods. We were able to categorize programs by the author's gender accurately by 87.5%.

In the third experiment we applied an attribute selection evaluator, information gain, to extract a subset of four features (as described in section 3.5.3). We then used the LOOCV technique to again create classification models using only four features. We were able to accurately predict the gender of 69.4% of our sample programs. In the third experiment we again used four features but applied the hold out technique to divide the dataset instead of the LOOCV technique (as described in section 3.5.3). We were able to accurately predict the gender of 67.3% of the programs.

In the fourth experiment we used the correlation feature subset attribute evaluator to identify a subset of seven features (as described in section 3.5.4). This gave better results than four features. We again used the hold out technique for dividing the dataset into training and testing sets. In the best case scenario we were able to correctly categorize 79.9% of the programs based on the gender. In the fifth experiment we used the seven features and the LOOCV technique (as described in section 3.5.3). With this combination we were able to accurately predict the gender of 71% of our sample programs.

We were interested in the results acquired from the first, the third, and the fifth experiment because we applied the LOOCV technique in each of these experiments. Use of the LOOCV technique is recommended over the hold out method to mitigate the risk of incorrect generalizations of the dataset [9, 18]. In addition the risk of random selection of data samples from one class was also reduced. Therefore, the models from these experiments should be able to generalize and extract more useful information from the given dataset in order to accurately classify future datasets.

After the application of machine learning techniques we performed further analysis of the subsets of features in the group of male-written and female-written programs. We were able to make some interesting observations based on the small dataset and various analyses.

1. From the subset of seven features we found that within our dataset of 100 programs a significant difference was found in the use of three features $/$, $>=$ and *double*. The

frequency of / and >= features was higher in male-written programs than female-written programs (as described in section 4.3). The frequency of the *double* feature was higher in female-written programs than in male-written programs.

2. We also explored relationships between pairs of features (as described in section 4.6). In our dataset / and + have a stronger positive linear relationship in male-written programs than in female-written programs.

3. Similarly, the features *bool* and == have a stronger positive linear relationship in male-written programs than in female-written programs.

During various analyses of our dataset we found potential threats including not enough data, few female authors, and varying problem domains. To solve these threats using machine learning techniques we oversampled female-written programs, we used LOOCV to mitigate the problem of incorrect generalization of dataset, and we considered various metrics and measures such as accuracy, TP, TN, FP, and f-measure.

5.1 Future Research Directions

Future work may be carried out in a number of directions. Some possible avenues are listed below.

- In the future it would be interesting to use another social variable such as the level of experience of our participants in various programming languages.

- In the future, we would like to perform statistical analyses of 50 features/attributes. In addition, we would carry out Student's t-test [5, 18] on the 50 features, the subset of features, and their frequency to identify the gender-based differences.

- It would be interesting to develop software to integrate all of the supervised learning algorithms that are used in this thesis to classify computer programs written in other programming languages.

- In the future we would be interested to use other attribute selection algorithms to identify the conservative estimates of the performance of the four classification models on the unseen large dataset.

- We currently have a small dataset composed of 100 data samples and a total of 2999 tokens. It would be interesting to extend this study by using a large dataset of programs written either in the same or different programming languages associated with male and female programmers. To handle the large dataset we will need to use other storage spaces rather than loading it into main memory as we are currently doing. For this purpose, we could use a tool like Hadoop [44] to distribute and parallelize the tasks of analyzing a large amount of data and to develop various classification models.

- In this work we used supervised learning methods to categorize computer programs written in the C++ programming language. It would be interesting to employ unsupervised learning methods [45] to identify groups of male and female programmers on the basis of the similarity among their coding styles in terms of their feature usages.

- Based on our exploratory effort, we now know that there are likely to be gender-based differences in programming. We would like to employ unsupervised learning methods and statistical techniques in order to cluster programs on the basis of the same problem domain, or identify groups of male and female programmers in terms of the similarities in their usage of features.

- We hope to use these results to improve teaching practice; to design the development teams; to further explore gender-based differences in programming; and to motivate more females to enter or stay in the field of computer science.

- The development of an Integrated Development Environment (IDE) may aid in providing suggestions of writing programs on the basis of gender. This way the quality

of software might be improved that is written by male and female programmers. This contribution might lead to the development of automatic error detection and correction if patterns of the group's most common mistakes are known.

Bibliography

[1] Part-of-speech tagger @ONLINE. http://cst.dk/tools/index.php.

[2] *SAS/STAT 9.2 User's Guide, 1:ANOVA-FREQ*, 4th edition, 2008.

[3] *IBM SPSS Statistics for Windows, Version 22.0*. Armonk, NY: IBM Corp., 2013.

[4] S. Argamon, J. Goulain, R. Horton, and M. Olsen. Vive la différence! text mining gender difference in french literature @ONLINE. *Digital Humanities Quarterly*, 3(2), 2009. http://www.digitalhumanities.org/dhq/vol/3/2/000042/000042.html.

[5] S. Argamon, M. Koppel, J. Fine, and A. R. Shimoni. Gender, genre, and writing style in formal written texts. *TEXT*, 23:321–346, 2003.

[6] A. Ben-hur and J. Weston. A users guide to support vector machines. *Methods in Molecular Biology*, 609:223–239, 2010.

[7] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[8] S. Burrows and S. M. M. Tahaghoghi. Source code authorship attribution using n-grams. *Proceedings of the 12th Australasian Document Computing Symposium, Melbourne, Australia, RMIT University*, pages 32–39, 2007.

[9] R. P. L. Buse and W. Weimer. Learning a metric for code readability. *IEEE Transactions on Software Engineering (TSE Special Issue on the ISSTA 2008 best papers)*, 36(4):546–558, 2010.

[10] H. Chen and M. Chau. Web mining: Machine learning for web applications. Technical report, University of Arizona.

[11] J. G. Cleary and L. E. Trigg. K*: An instance-based learner using an entropic distance measure. In *12th International Conference on Machine Learning*, pages 108–114, 1995.

[12] L. M. Connelly. t-tests. *Medsurg Nursing*, 20(6):341, 2011.

[13] H. C. Currie. A projection of socio-linguistics: the relationship of speech to social status. *The Southern Speech Journal*, 18:28–37, 1952.

[14] P. Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

[15] J. L. Fischer. Social influences on the choice of a linguistics variant. *Word*, 1958.

[16] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine learning*. Addison-Wesley Publishing Company, Inc.

[17] M. A. Hall. *Correlation-based Feature Subset Selection for Machine Learning*. PhD thesis, University of Waikato, 1998.

[18] J. Han, M. Kamber, and J. Pei. *Data Mining Concepts and Techniques*. Elsevier and Morgan Kaufmann Publishers, 3rd edition, 2012.

[19] P. Hart. The condensed nearest neighbour rule. *IEEE Trans. Inf. Theory.*, 14(3):515–516, 1968.

[20] G. Holmes, A. Donkin, and I. H. Witten. Weka: A machine learning workbench. In *Proc. Australia and New Zealand Conf. Intelligent Information Systems*, 1994.

[21] C. Hsu, C. Chang, and C. Lin. A practical guide to support vector classification, 2010.

[22] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. Technical report, Universitá Dortmund.

[23] T. Joachims. Making large-scale svm learning practical. advances in kernel methods-support vector learning. 1999.

[24] G. H. John and P. Langley. Estimating continuous distributions in bayesian classifiers. In *Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345, 1995.

[25] S. M. Kamruzzaman, F. Haider, and A. R. Hasan. Text classification using data mining. *Proc. International Conference on Information and Communication Technology in Management (ICTM-2005)*, 2005.

[26] J. Kivinen and M. K. Warmuth. Additive versus exponentiated gradient updates for linear prediction. *Information and Computation*, 132(1):1–64, 1997.

[27] M. Koppel, S. Argamon, and A. Shimoni. Automatically categorizing written texts by author gender. *Literary and Linguistic Computing*, 17(4):401–412, 2002.

[28] I. Krsul and E. Spafford. Authorship analysis: Identifying the author of a program. *Computers and Security*, 16(3):233–248, 1997.

[29] L. Kukolich and R. Lippmann. *LNKnet User's Guide*. MIT Lincoln Laboratory, MIT Technology Licensing Office, RM E32-300, 200 Carleton Street, Cambridge, MA 02142-1324, 1995.

[30] M. Kukreja, S. A. Johnston, and P. Stafford. Comparative study of classification algorithms for immunosignaturing data. *BMC Bioinformatics*, 2012.

[31] W. Labov. The linguistic variable as a structural unit. *Washington Linguistics Review 3*, pages 4–22, 1966.

[32] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linearthreshold algorithm. *Machine Learning*, 2(4):285–318, 1987.

[33] R. Mamoun and M. A. Ahmed. A comparative study on different types of approaches to the arabic text classification. 2014.

[34] L. D. Misek-Falkoff. The new field of software linguistics: An early-bird view. *SIGMETRICS performance evaluation review*, 11(2):35–51, 1982.

[35] W. O'Grady and J. Archibald. *An Introduction Contemporary Linguistic Analysis*. Pearson Education Canada, 6th edition, 2008.

[36] J. F. Pane, C. A. Ratanamahatana, and B. A. Myers. Studying the language and structure in non-programmers' solutions to programming problems. *International Journal of Human-Computer Studies*, 54(2):237–264, 2001.

[37] P. Pavlidis, I. Wapinski, and W. S. Noble. Support vector machine classification on the web. 20(4):586–587, 2004.

[38] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[39] J. E. Rice, I. Genee, and F. Naz. Linking linguistics and programming: How to start?(work in progress). In *Proc. 25th Annual Psychology of Programming Interest Group Conference-PPIG*, 2014.

[40] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 5(24), 1988.

[41] N. Sebe, M. S. Lew, Y. Sun, I. Cohen, T. Gevers, and T. S. Huang. Authentic facial expression analysis. *Image and Vision Computing*, 25(12):1856–1863, 2007.

[42] S. Vijayarani and M. Muthulakshmi. Comparative analysis of bayes and lazy classification algorithms. *International Journal of Advanced Research in Computer and Communication Engineering*, 2(8), 2013.

[43] R. Wardhaugh. *An Introduction to Sociolinguistics*. Blackwell Publishers, Oxford, UK, 1992.

[44] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2009.

[45] I. H. Witten and E. Frank. *Data Mining Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers, 2nd edition, 2005.

[46] G. Zweig, P. Mguyen, J. Droppo, and A. Acero. Continuous speech recognition with a tf-idf acoustic model. *International Speech Communication Association*, 2010.

Appendix A

Terminology

Here we provide a brief overview of the terminology that is used throughout this thesis, which may be unfamiliar to non-programmers.

- **Program:**
  A program is a text file created by a programmer using a known programming (or artificial) language such as C, C++, or Java. A program may contain variables, instructions, functions, loops, and other statements.

- **Software:**
  Software is a computer program composed of either a single file or a number of distinct files written in any programming language to fulfill specific computational tasks. In professional settings, teams of people work towards the development of the same software. For this reason, the term "software" refers to data and associated documentation such as system documentation, user documentation and so forth.

- **Open-Source Software:**
  Open-source software is a kind of software which has its program files, data, and documentation(s) publicly available. The program files may be enhanced or modified by other programmers for their own and others' usage. For example, WEKA, which is used in this thesis, is an open-source software.

- **Feature/Attribute:**
  In order to categorize any kind of dataset, WEKA and SVM^{light} require a feature list. A feature is created by transforming raw data into a representation that is suited for machine learning tasks. In the case of computer program files (as a dataset), keywords, operators, loops (for/while), brackets, and comments, are considered as features. The term "feature" is interchangeable with "attribute".

- **Feature Vector:**
  A set of features that represent a single computer program is referred to as a feature vector. The format <feature>:<value> is used, where <feature> is the index of an attribute which starts either from one or zero, and <value> represents the frequency of occurrence of a feature in terms of tf-idf score.

- **Data Tuple/Instance:**
  A data tuple represents a single file, which is a component of a given dataset. Gen-

erally, a data tuple is composed of class labels and feature vectors. The term “instances”,“samples”, and “examples” are used interchangeably with tuple.

A.1 Project Survey and Questionnaire

Project Information Sheet:
Sociolinguistics in Computer Programming

I would like to invite you to be a participant in a study combining the fields of linguistics and computer science. This study is taking place under the supervision of Dr. Jackie Rice who is presently serving as the Associate Dean and Associate Professor at the Dept. of Math and Computer Science. I am aiding Dr. Jackie Rice in this study as a research assistant.

The purpose of the study is to investigate how analytic tools from the field of linguistics might be applied to samples of computer programs. We hope this will allow us to identify how information about the programmers or their sociological characteristics might be used to improve current approaches to the development of computer programs. In addition, information from this work may also be used in improving computer science education practices, in much the way applied linguistics has been used in teaching English as a second language.

Should you agree to participate your participation in this study will consist of permitting us to examine and analyse the C++ code that you have written for your assignment(s) in the previous computer science courses by providing us the complete source code. You need to sign the attached consent form and complete a short follow-up questionnaire. The questionnaire should take less than 10 minutes to complete and is attached to this letter.

Your name will be removed from the data gathered during this study, and the data gathered will be used only for research and teaching publication purposes. There are no anticipated risks to this study. Anonymised data will be stored on a secure password-protected system to maintain participants' confidentiality, and all those with access to your data will sign confidentiality agreements. Although there is no direct benefit to the participants, information from this study will be used to advance the field of computer science.

Participation is entirely voluntary and you may withdraw at any time. Should you wish to withdraw the data you have submitted to us will be destroyed. To withdraw from the study, ask any further questions about the study, or to request an executive summary before the results are published please contact Dr. Jackie or me:

Dr. Jackie Rice

j.rice@uleth.ca

Associate Dean and Associate Professor,

Dept. of Math and Computer Science,

University of Lethbridge, Lethbridge, AB, Canada

Fariha Naz

fariha.naz@uleth.ca

Dept. of Math and Computer Science,

University of Lethbridge, Lethbridge, AB, Canada

Questions regarding your rights as a participant in this research may be addressed to the Office of Research Services, University of Lethbridge (Phone: 403-329-2747 or Email: research.services@uleth.ca).

CONSENT

YES / NO I have read and understood the information sheet.

YES / NO I understand the nature of this study and agree to participate.

YES / NO I understand that I can withdraw at any time.

Signed _____ Date _____

(Questionnaire is on the following page)

QUESTIONNAIRE

First Name: _____

Last Name: _____

Gender: _____

(used to match questionnaire information with your code submission)

1. What was your first language spoken at home?

Answer: _____

2. What was your major during your undergraduate degree (e.g. Electronics)? Currently, what is your major in your graduate program (e.g. Computer Science, MSc)?

Answer:

- **Undergraduate Major-** _____
- **Graduate Major-** _____

3. What was the first computer programming language that you learned?

Answer: _____

4. What was the second computer programming language that you learned?

Answer: _____

5. Overall how many years/months of experience would you say you have as a programmer?

Answer: _____

6. For how long you have been developing computer programs in the C++ programming Language? (e.g. 1 year, 2 years, 5 years, 10 years, or longer than 10 years).

Answer: _____

7. What other programming languages do you consider yourself to be familiar with (e.g. for writing moderately complex programs)?

Answer:

- _____
- _____

(continued on the following page)

≅ _____

≅ _____

8. If you have worked with several programming languages, how long have you worked with each, either as a professional or a hobbyist (e.g. Java: 4 years)?

Answer:

≅ _____

≅ _____

≅ _____

≅ _____

9. While writing the code that you have submitted for this study, what was your predominant consideration (e.g. completion of assignment)?

Answer: _____

10. Considering your answer to question 9 above, how do you feel that this was reflected in your code (e.g. able to complete assignment/your code is similar to what you had in mind)?

Answer: _____

11. Did you work alone on this code, or in a group/partner situation?

Answer: _____

12. If this was a joint/group effort, how much of the code would you estimate was "yours"? For instance, 10%, 50%, 75%?

Answer: _____

13. Do you expect to be working with this code again?

Answer: _____

End of questionnaire. Thank-you!

Appendix B

Detail of Features

Here we provide the list of features with their descriptions taken from the following resources [39]:

1. Keywords Detail: `http://en.cppreference.com/w/cpp/keyword`
2. Operators Detail: `http://en.cppreference.com/w/cpp/language/operator_precedence`
3. Comments Detail: `http://en.cppreference.com/w/cpp/comment`

Table B.1: C++ Reserved Keywords and their Meanings.

Description	Keywords
Loops	*for*, *while*
Datatypes	*int*, *float*, *char*, *double*, *bool*
Constants	*const*
Preprocessors	*#define* , *#include*
Access labels	*public*, *private*
Standard Input (statement)	*cin*
Decision making	*switch*
Built-in functions	*return*, *exit*
Standard Output (statements)	*cout*
Standard Error or Output (statements)	*cerr*
To include namespace	*using*
Dynamic memory allocation	*new*
Functions with no return value	*void*

Table B.2: C++ Operators and their Details.

Description	Operator
Equality	==
Inequality	!=
Less than	<
Division	/
Addition	+
Increment	++
Decrement	--
Address of	&
Logical not	!
Subtraction	-
Assignment	=
Bitwise OR	|
Logical OR	||
Greater than	>
Logical AND	&&
Multiplication	*
Member selection	->
Division assignment	/=
Less than or equal to	<=
Addition assignment	+=
Greater than or equal to	>=
Subtraction assignment	-=
Multiplication assignment	*=

Table B.3: C++ Comments

Description	Comments
Multi-line	/* */, //*
Single-line	//

Appendix C

Frequency of Features

Table C.1: Frequency of Features in Female-Written Programs.

Programs	/	==	+	>=	*double*	*char*	*bool*	**Total Tokens**
1	0	46	2	0	0	5	1	1288
2	7	3	64	0	2	2	1	534
3	3	0	7	0	0	2	6	414
4	3	0	0	0	11	2	1	613
5	3	6	14	0	4	4	0	771
6	4	16	9	0	0	0	2	1449
7	2	4	11	0	2	5	0	647
8	0	5	2	0	0	3	0	458
9	1	7	14	0	2	15	0	836
10	2	0	2	0	22	1	0	439
11	4	43	6	3	0	1	7	1150
12	0	5	1	0	0	2	1	570
13	0	0	1	0	0	3	1	1421
14	0	0	0	0	0	2	1	474
15	12	3	10	0	6	2	2	868
16	0	1	1	0	0	3	2	439
17	2	0	2	0	17	0	0	578
18	2	0	3	0	34	0	0	377
19	4	42	6	1	0	0	6	956
20	1	5	1	2	0	4	2	468
21	4	16	6	0	0	0	8	1039

22	2	1	0	0	17	1	0	573
23	0	16	23	0	0	3	9	**2950**
24	3	0	7	0	1	2	7	570
25	2	0	3	0	34	0	0	526
26	0	1	0	0	0	5	2	411
27	2	5	16	0	2	11	0	621
28	2	6	8	3	2	4	0	652
29	0	0	0	0	19	0	0	332
30	0	1	0	0	0	3	2	198
31	4	12	7	0	0	0	6	805
32	3	14	1	0	2	2	2	1601
33	2	3	8	2	11	9	0	672
34	0	1	0	0	0	3	3	221
35	2	6	1	0	0	0	2	536
36	2	0	3	2	36	0	0	359
37	0	1	5	0	0	3	6	297
38	4	55	6	3	0	0	9	901
39	0	4	21	0	0	2	4	1364
40	1	0	17	0	0	2	1	1223
41	0	0	0	0	0	0	0	244
42	2	0	0	0	0	0	0	146
43	0	9	14	0	0	0	4	805
44	0	6	5	3	0	0	3	959
45	3	0	0	1	0	0	0	306
46	0	0	1	0	0	0	0	**141**
47	0	2	0	0	0	2	3	210
48	2	0	0	0	17	0	0	535
49	0	7	6	0	0	3	5	410
50	0	0	4	2	0	18	0	249
Total	90	352	318	22	241	129	109	34607

Average	1.8	7.04	6.36	0.44	4.82	2.58	2.18	692.14

Table C.2: Frequency of Features in Male-Written Programs.

Programs	/	==	+	>=	*double*	*char*	*bool*	**Total Tokens**
1	0	1	0	0	0	0	0	**107**
2	2	11	15	0	0	3	0	464
3	0	9	9	5	0	1	0	557
4	0	0	0	0	0	2	0	334
5	0	8	0	0	0	2	1	570
6	12	6	1	0	0	2	2	584
7	3	0	0	0	3	2	2	421
8	0	14	0	2	0	2	0	1250
9	4	0	0	0	2	2	2	522
10	1	4	11	2	0	2	2	692
11	0	3	5	0	1	2	4	665
12	0	2	3	0	0	2	0	643
13	3	1	2	0	4	2	1	1166
14	30	8	48	2	0	2	0	**2999**
15	1	4	2	0	0	0	1	246
16	1	3	3	2	0	0	1	479
17	1	3	5	3	0	0	3	505
18	1	4	3	8	0	0	1	611
19	1	3	5	3	0	0	1	298
20	4	17	6	0	0	0	10	697
21	1	3	3	1	0	1	1	536
22	3	29	6	0	0	0	7	1210
23	4	43	6	0	0	1	6	854
24	0	1	4	0	0	4	4	262
25	8	45	12	3	0	1	8	1774
26	8	33	12	0	0	1	6	1083

27	8	44	12	2	0	0	6	993
28	4	19	6	0	0	1	7	1234
29	4	5	14	0	2	5	0	745
30	2	5	7	0	0	5	0	534
31	2	3	8	2	14	5	0	637
32	2	3	7	0	0	11	0	463
33	2	3	12	0	0	5	0	603
34	10	3	11	4	29	5	0	1104
35	2	3	5	0	6	5	0	650
36	4	7	3	0	0	5	0	622
37	0	4	0	0	0	3	0	244
38	0	2	0	0	0	0	0	170
39	4	18	6	0	0	0	7	879
40	4	18	6	0	0	1	6	965
41	4	17	7	0	0	1	6	1077
42	0	1	2	0	0	0	2	272
43	2	7	6	3	0	0	2	741
44	3	29	6	0	0	0	7	742
45	1	8	3	1	0	0	1	490
46	8	44	12	3	0	1	8	1510
47	4	10	7	3	0	1	6	594
48	5	24	6	0	0	0	12	1088
49	1	2	1	0	0	0	2	334
50	4	10	7	3	0	1	6	677
Total	168	544	315	52	61	89	141	36897
Averages	3.36	10.88	6.3	1.04	1.22	1.78	2.82	737.94

Table C.3: Frequency of Features Per 100 Tokens in Female-Written Programs.

Programs	/	==	+	>=	*double*	*char*	*bool*
1	0	3.571	0.155	0	0	0.388	0.077
2	1.31	0.561	11.985	0	0.374	0.374	0.187
3	0.724	0	1.69	0	0	0.483	1.449
4	0.489	0	0	0	1.794	0.326	0.163
5	0.389	0.778	1.815	0	0.521	0.518	0
6	0.276	1.104	0.621	0	0	0	0.138
7	0.309	0.618	1.7	0	0.309	0.772	0
8	0	1.091	0.436	0	0	0.655	0
9	0.119	0.837	1.674	0	0.239	1.794	0
10	0.455	0	0.455	0	5.011	0.227	0
11	0.347	3.739	0.521	0.26	0	0.086	0.608
12	0	0.877	0.175	0	0	0.35	0.175
13	0	0	0.07	0	0	0.211	0.07
14	0	0	0	0	0	0.421	0.21
15	1.382	0.345	1.152	0	0.691	0.23	0.23
16	0	0.227	0.227	0	0	0.683	0.455
17	0.346	0	0.346	0	2.941	0	0
18	0.53	0	0.795	0	9.018	0	0
19	0.418	4.393	0.627	0.104	0	0	0.627
20	0.213	1.068	0.213	0.427	0	0.854	0.427
21	0.384	1.539	0.577	0	0	0	0.769
22	0.349	0.174	0	0	2.966	0.174	0
23	0	0.542	0.779	0	0	0.101	0.305
24	0.526	0	1.228	0	0.175	0.35	1.228
25	0.38	0	0.57	0	6.463	0	0
26	0	0.243	0	0	0	1.216	0.486
27	0.322	0.805	2.576	0	0.322	1.771	0
28	0.306	0.92	1.226	0.46	0.306	0.613	0
29	0	0	0	0	5.722	0	0

30	0	0.505	0	0	0	1.515	1.01
31	0.496	1.49	0.869	0	0	0	0.745
32	0.187	0.874	0.062	0	0.124	0.124	0.124
33	0.297	0.446	1.19	0.297	1.636	1.339	0
34	0	0.452	0	0	0	1.357	1.357
35	0.373	1.119	0.186	0	0	0	0.373
36	0.557	0	0.835	0.557	10.027	0	0
37	0	0.336	1.683	0	0	1.01	2.02
38	0.443	6.104	0.665	0.332	0	0	0.998
39	0	0.293	1.539	0	0	0.146	0.293
40	0.081	0	1.39	0	0	0.163	0.081
41	0	0	0	0	0	0	0
42	1.369	0	0	0	0	0	0
43	0	1.118	1.739	0	0	0	0.496
44	0	0.625	0.521	0.312	0	0	0.312
45	0.98	0	0	0.326	0	0	0
46	0	0	0.709	0	0	0	0
47	0	0.952	0	0	0	0.952	1.428
48	0.373	0	0	0	3.177	0	0
49	0	1.707	1.463	0	0	0.731	1.219
50	0	0	1.606	0.803	0	7.228	0
Average	0.294	0.789	0.921	0.077	1.036	0.543	0.361

Table C.4: Frequency of Features Per 100 Tokens in Male-Written Programs.

Programs	/	==	+	>=	*double*	*char*	*bool*
1	0	0.934	0	0	0	0	0
2	0.431	2.37	3.232	0	0	0.646	0
3	0	1.615	1.615	0.897	0	0.179	0
4	0	0	0	0	0	0.598	0
5	0	1.403	0	0	0	0.35	0.175
6	2.054	1.027	0.171	0	0	0.342	0.342
7	0.712	0	0	0	0.712	0.475	0.475
8	0	1.12	0	0.16	0	0.16	0
9	0.766	0	0	0	0.383	0.383	0.383
10	0.144	0.578	1.589	0.289	0	0.289	0.289
11	0	0.451	0.751	0	0.15	0.3	0.601
12	0	0.311	0.466	0	0	0.311	0
13	0.257	0.085	0.171	0	0.343	0.171	0.085
14	1	0.266	1.6	0.066	0	0.066	0
15	0.406	1.626	0.813	0	0	0	0.406
16	0.208	0.626	0.626	0.417	0	0	0.208
17	0.198	0.594	0.99	0.594	0	0	0.594
18	0.163	0.654	0.49	1.309	0	0	0.163
19	0.335	1.006	1.677	1.006	0	0	0.335
20	0.573	2.439	0.86	0	0	0	1.434
21	0.186	0.559	0.559	0.186	0	0.186	0.186
22	0.247	2.396	0.495	0	0	0	0.578
23	0.468	5.035	0.702	0	0	0.117	0.702
24	0	0.381	1.526	0	0	1.526	1.526
25	0.45	2.536	0.676	0.169	0	0.056	0.45
26	0.738	3.047	1.108	0	0	0.092	0.554
27	0.805	4.431	1.208	0.201	0	0	0.604
28	0.324	1.539	0.486	0	0	0.081	0.567
29	0.536	0.671	1.879	0	0.268	0.671	0

30	0.374	0.936	1.31	0	0	0.936	0
31	0.313	0.47	1.255	0.313	2.197	0.784	0
32	0.431	0.647	1.511	0	0	2.375	0
33	0.331	0.497	1.99	0	0	0.829	0
34	0.905	0.271	0.996	0.362	2.626	0.452	0
35	0.307	0.461	0.769	0	0.923	0.769	0
36	0.643	1.125	0.482	0	0	0.803	0
37	0	1.639	0	0	0	1.229	0
38	0	1.176	0	0	0	0	0
39	0.455	2.047	0.682	0	0	0	0.796
40	0.414	1.865	0.621	0	0	0.103	0.621
41	0.371	1.578	0.649	0	0	0.092	0.557
42	0	0.367	0.735	0	0	0	0.735
43	0.269	0.944	0.809	0.404	0	0	0.269
44	0.404	3.908	0.808	0	0	0	0.943
45	0.204	1.632	0.612	0.204	0	0	0.204
46	0.529	2.913	0.794	0.198	0	0.066	0.529
47	0.673	1.683	1.178	0.505	0	0.168	1.01
48	0.459	2.205	0.551	0	0	0	1.102
49	0.299	0.598	0.299	0	0	0	0.598
50	0.59	1.477	1.033	0.443	0	0.147	0.886
Averages	0.379	1.322	0.815	0.154	0.152	0.315	0.378

In the following Tables, we present two correlation matrix computed for males and females. This matrix is symmetric, so the diagonal values in the matrix are always 1. These values represent that there is a strong relationship between the same features which is of no interest. The values above and below the diagonal are the same. Thus, we report the values only above the diagonal.

Table C.5: Correlation Based on Raw Frequency of Features in Female-Authored Programs.

Features	/	==	+	>=	*double*	*char*	*bool*
/	1	0.206	0.356	0.072	0.094	-0.18	0.145
==		1	0.05	0.395	-0.257	-0.086	0.535
+			1	-0.057	-0.148	0.134	0.129
>=				1	-0.014	0.118	0.152
double					1	-0.206	-0.389
char						1	-0.226
bool							1

Table C.6: Correlation Based on Raw Frequency of Features in Male-Authored Programs.

Features	/	==	+	>=	*double*	*char*	*bool*
/	1	0.304	0.817	0.11	0.151	0.028	0.17
==		1	0.244	0.05	-0.175	-0.298	0.73
+			1	0.195	0.072	0.136	0.052
>=				1	0.212	-0.183	-0.063
double					1	0.344	-0.214
char						1	-0.437
bool							1